WIMOAIR. MONITORIZACIÓN REMOTA DE CALIDAD DEL AIRE

WIMOAIR. WIRELESS MONITORIZATION OF AIR QUALITY

TRABAJO FIN DE MÁSTER
CURSO 2020-2021

AUTOR
VÍCTOR GOICOECHEA ENRIQUE

DIRECTORES
JOSÉ IGNACIO GÓMEZ PÉREZ
FRANCISCO DANIEL IGUAL PEÑA

CURSO: 2020/2021
MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

WIMOAIR. MONITORIZACIÓN REMOTA DE CALIDAD DEL AIRE

WIMOAIR. WIRELESS MONITORIZATION OF AIR QUALITY

Trabajo de Fin de Máster en Internet de las Cosas
Departamento de Arquitectura de Computadores y Automática

Autor
Víctor Goicoechea Enrique

DIRECTORES
José Ignacio Gómez Pérez
Francisco Daniel Igual Peña

Convocatoria: Septiembre 2021
Calificación: 7.5

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

24 de SEPTIEMBRE de 2021

Dedicatoria

A mi familia por estar siempre
apoyándome en estos momentos tan
difíciles y ser los que me dan fuerzas para
seguir adelante.

Agradecimientos

Quiero agradecer a mis profesores del TFM tanto a Nacho como a Fran, por haber realizado bajo su dirección, un trabajo muy interesante de cara al control de la calidad del aire.

Resumen

Con el paso del tiempo los sistemas de información han ido siendo mejorados, haciendo posible su integración en lugares donde pasan desapercibidos por las personas. Por ejemplo, a la hora de coger el metro, conducir un coche o ir al supermercado, entre otras actividades. Todo ello proporcionará, una gran fuente de datos, la cuál es tratada para mejorar la vida de las personas, haciéndola más cómoda y sencilla.

El objetivo del proyecto es brindar una solución para la medición de la calidad del aire en espacios cerrados como pueden ser salas de cine, sucursales bancarias, aulas o centros médicos, entre otros. En particular, este trabajo se enfocará en recopilar y visualizar los datos de mediciones de niveles de CO2, TVOC[1], temperatura y humedad, así como el aforo actual. De esta manera, se podrá conocer con precisión la calidad del aire de los espacios cerrados.

Este proyecto tiene una especial relevancia ya que trata de mitigar los efectos causados por el contexto actual de pandemia, donde la calidad del aire en espacios cerrados influye en aumentar o disminuir las probabilidades de contagio por COVID-19.

Para abordar esta problemática se desarrolló un software pensado para trabajar sobre determinados componentes hardware, que permitan su despliegue y uso en cualquier lugar. Este sistema posee las siguientes funcionalidades:

- Medir los niveles de CO2, TVOC, temperatura y humedad y enviarlos vía MQTT con transporte SSL/TLS a una Raspberry Pi.
- Detección de personas para saber el aforo actual de un espacio cerrado, enviado la información vía MQTT con transporte SSL/TLS.
- Almacenamiento en una base de datos de series temporales de las mediciones enviadas vía MQTT con transporte SSL/TLS a un mini servidor de Raspberry Pi.

[1] Sirve para medir la concentración de compuestos orgánicos volátiles (TVOC, son sus siglas en inglés), que se convierten fácilmente en vapores o gases, contribuyendo a la mala calidad del aire. Estos compuestos son liberados por la quema de combustibles, disolventes y pinturas, entre otros.

- Visualización de los datos para conocer la calidad del aire en tiempo real de un espacio cerrado.
- Sistema de gestión de alarmas para enviar avisos a través de un bot en Telegram y poder informar sobre posibles riesgos.

Palabras clave

- ESP32
- Si7021
- SGP30
- MQTT
- Raspberry Pi
- Internet de las cosas
- Visión por computación
- Docker
- Visualización de datos
- Monitorización de calidad del aire

Abstract

With the passage of time, information systems have been improved, making it possible to integrate them in places where they go unnoticed by people. For example, when taking the subway, driving a car or going to the supermarket, among other activities. All this will provide a great source of data, which is treated to improve people's lives, making it more comfortable and simple.

The objective of the project is to provide a solution for measuring air quality in closed spaces such as movie theaters, bank branches, classrooms or medical centers, among others. In particular, this work will focus on collecting and visualizing data from measurements of CO2 levels, TVOC[2], temperature and humidity, as well as the current capacity. In this way, the quality of the air in enclosed spaces can be accurately known.

This project has a special relevance since it tries to mitigate the effects caused by the current pandemic context, where the quality of the air in closed spaces influences to increase or decrease the chances of contagion by COVID-19.

To tackle this problem, we propose a holistic software solution running on low-power processors, which allows an easy deployment and ubiquity. This system features the following functionalities:

- Measure CO2, TVOC, temperature and humidity levels and send them via MQTT with SSL / TLS transport to a centralized server (in our case, a Raspberry Pi).
- Detection of people to know the current occupation of a closed space, transmitting it via MQTT with SSL / TLS transport.
- Storage in a time series database of measurements sent via MQTT with SSL / TLS transport to a centralized server.
- Visualization of the data in an easy-to-use dashboard, that allows an easy observation of the current status of the target facility.
- Alarm management system to send notifications through a Telegram bot and to be able to report under certain circumstances that reveal possible health risks.

[2] It is used to measure the concentration of volatile organic compounds (TVOC), which are easily converted to vapors or gases, contributing to poor air quality. These compounds are released by the burning of fuels, solvents and paints, among others.

Keywords

- ESP32
- Si7021
- SGP30
- MQTT
- Raspberry Pi
- Internet of things
- Computer vision
- Docker
- Data visualization
- Air quality monitoring

Índice

1. Introducción

1.1. Motivación

El IoT (**I**nternet **o**f **T**hings) o Internet de las cosas es la interconexión digital en tiempo real de dispositivos cotidianos con Internet, lo cual permite recopilar y compartir información con otros dispositivos o centros de control.

Para lograr la recopilación de datos, los sensores juegan un papel crucial, ya que están integrados en todo tipo de dispositivo físico. De esta manera, podrán estar en un dispositivo móvil, electrodomésticos, vehículos, señales de tráfico y casi cualquier objeto que se pueda encontrar en el día a día. Estos sensores generan grandes cantidades de datos, que se volcarán en plataformas IoT.

Las plataformas IoT se encargan de procesar, almacenar y analizar los grandes volúmenes de datos para compartir los resultados y/o predicciones. Esto permite ofrecer una mejor experiencia de usuario o hacer más eficientes procesos de producción.

Las aplicaciones para IoT son casi ilimitadas. Aquí se muestran algunas de ellas:

- S*mart farms*, que permiten al agricultor ser más eficiente en sus cultivos, ahorrando agua con un riego inteligente o conduciendo de forma autónoma los tractores.
- Smart factories, más conocido con el nombre de IIoT (**I**ndustrial **I**nternet **o**f **T**hings), que servirá para ayudar en la monitorización de productos en tiempo real, gestión de inventario o control de flujos de producción.
- Smart home, donde se dará la capacidad de controlar los electrodomésticos con comandos de voz o desde otro dispositivo como puede ser un smartphone.

Como se puede ver, IoT es usado para resolver problemas de la vida real. Un problema actual es el contexto de pandemia por COVID-19, el cual se trata de la irrupción de un nuevo virus que se transmite por el aire y ha tenido una rápida propagación por el mundo.

En este proyecto se pretende dar una solución IoT para disminuir la propagación del COVID-19, mediante el control de la calidad del aire en espacios cerrados, ya que el

aire es el medio principal de propagación del COVID-19 y si se mantiene una correcta calidad del aire se puede disminuir la capacidad de su transmisión en espacios cerrados.

1.2. Objetivos

El objetivo principal del proyecto es desarrollar un sistema capaz de recopilar, almacenar y visualizar datos de mediciones de CO2, TVOC[3], temperatura, humedad y aforo para poder medir la calidad del aire en un espacio cerrado. También, el sistema será capaz de enviar una alerta al teléfono móvil, si se supera un determinado nivel de CO2, temperatura o aforo.

Para ello este objetivo se divide a su vez en objetivos más específicos:

- Desarrollar un software que permita a un nodo sensor gestionar y enviar las mediciones de los niveles de CO2, TVOC, temperatura y humedad. De esta manera, se podrá conocer la calidad del aire en un espacio cerrado.
- Elaborar un sistema capaz de medir el aforo dentro de un espacio cerrado. La solución que se desarrollará, será capaz de detectar personas mediante el uso de un modelo de detección de objetos, a través de una cámara.
- Desarrollar un sistema capaz de procesar, almacenar y visualizar los datos que envíen la medición de calidad del aire y la medición del aforo. Algunas de sus funcionalidades serán:
    - Recibir los datos enviados por los nodos sensores (medidor de calidad del aire y del aforo), vía MQTT con transporte SSL/TLS, utilizando el bróker MQTT de eclipse-mosquitto.
    - Gestionar las interacciones entre las diferentes aplicaciones haciendo uso de Node Red.
    - Almacenar la gran cantidad de datos recibidos en InfluxDB, siendo esta una base de datos de series temporales.

[3] Sirve para medir la concentración de compuestos orgánicos volátiles (TVOC, son sus siglas en inglés), que se convierten fácilmente en vapores o gases, contribuyendo a la mala calidad del aire. Estos compuestos son liberados por la quema de combustibles, disolventes y pinturas, entre otros.

- Visualizar los datos almacenados, en un panel de control, utilizando Grafana.
- Almacenar en MongoDB la información necesaria sobre los usuarios que se suscriban a las alertas en el bot de Telegram para poder enviar alarmas cuando se sobrepasen determinados límites que indican una mala calidad del aire.

1.3. Plan de trabajo

Para la consecución de los anteriores objetivos se desarrollaron las siguientes tareas:

- Investigación sobre el dispositivo ESP32. En esta primera tarea se buscó información acerca de cómo desarrollar software para poder enviar los datos vía MQTT con cifrado TLS y cómo comunicar el ESP32 con los sensores Si7021 y SGP30, para llevar a cabo las tareas de medición de los niveles de CO2, TVOC, temperatura y humedad.
- Investigación sobre software desarrollado para la detección de objetos. En concreto, para detectar personas en una imagen, donde se encontró el código desarrollado por Google Coral, permitiendo obtener el aforo aproximado de un espacio cerrado.
- Diseño y desarrollo de un sistema para el procesamiento, almacenamiento y visualización de los datos. Se buscó información acerca de Docker y cómo esta herramienta podría ayudar a cubrir las necesidades de desplegar servicios capaces de procesar, almacenar y visualizar los datos sin dar problemas de compatibilidad en la instalación de aplicaciones y poder ser desplegado de manera rápida en cualquier dispositivo capaz de soportar Docker.
- Pruebas sobre el sistema. Esta tarea final, consistió en realizar diferentes evaluaciones del sistema, como ver si el panel de control visualizaba en tiempo real las mediciones aportadas por los sensores y comprobar él envió de mensajes de alerta al bot de Telegram cuando se superan determinados niveles de CO2, temperatura o aforo.

2. Estado de la cuestión

En este capítulo se presentará el estado de la cuestión vinculado al desarrollo del proyecto. Concretamente, se dará una visión general de la estructura del sistema, donde se verán los nodos que lo conforman. También se incluirán proyectos similares que resuelven los problemas de medir la calidad del aire, el aforo y la monitorización de los datos.

2.1. Esquema general del sistema

En la figura 2.1, se puede observar que el sistema está compuesto por 3 componentes principales:

- Nodo que mide la calidad del aire.
- Nodo que mide el aforo.
- Mini servidor que monitoriza los datos de los diferentes nodos, el cual utilizará Docker para desplegar servicios que ayudarán a gestionar los contenedores que procesarán, almacenarán y visualizarán los datos.

El sistema que se desarrolló es capaz de recopilar los datos de las mediciones de calidad del aire y aforo mediante el mini servidor. Los nodos sensores podrán ser colocados en distintos espacios cerrados y serán procesados por el mini servidor que hará de centro de datos para el procesamiento, almacenamiento y visualización en tiempo real de los datos enviados por los nodos. Para lograr esto, el mini servidor utilizará la tecnología Docker para desplegar los siguientes servicios:

- **Portainer**, será un servicio que usará una página web para facilitar la gestión del resto de servicios del mini servidor de manera intuitiva y sin tener que tocar una consola de comandos.
- **Eclipse-mosquitto**, este servicio se encargará de la gestión del bróker MQTT, protocolo de comunicación utilizado para que los nodos sensores envíen la información de las mediciones por Internet al mini servidor.
- **InfluxDB**, este servicio será usado para almacenar la información recibida de los nodos sensores y poder consultarla para su análisis y visualización en un panel de control.

- **Grafana**, este servicio se encargará de conectarse con InfluxDB y visualizar, de una forma amigable e intuitiva, los datos más relevantes de las mediciones de los nodos sensores. Este servicio podrá ser editado por el usuario para adaptarlo a sus necesidades, donde mostrará los datos que considere más interesantes. En el proyecto se entregará, un ejemplo de panel de control, donde se mostrarán los niveles de CO2, temperatura y aforo.
- **MongoDB**, este servicio será usado para almacenar un identificador de los usuarios que se suscriban al sistema de alertas del bot de Telegram encargado de notificar las alarmas y las suscripciones al sistema de alertas.
- **Node Red**, este servicio será el encargado de recibir los datos por MQTT y procesarlos para almacenarlos en InfluxDB y MongoDB. Este servicio también se encargará de la gestión del sistema de alarmas, tanto para el envío de alertas como para la gestión de las suscripciones al bot de Telegram.

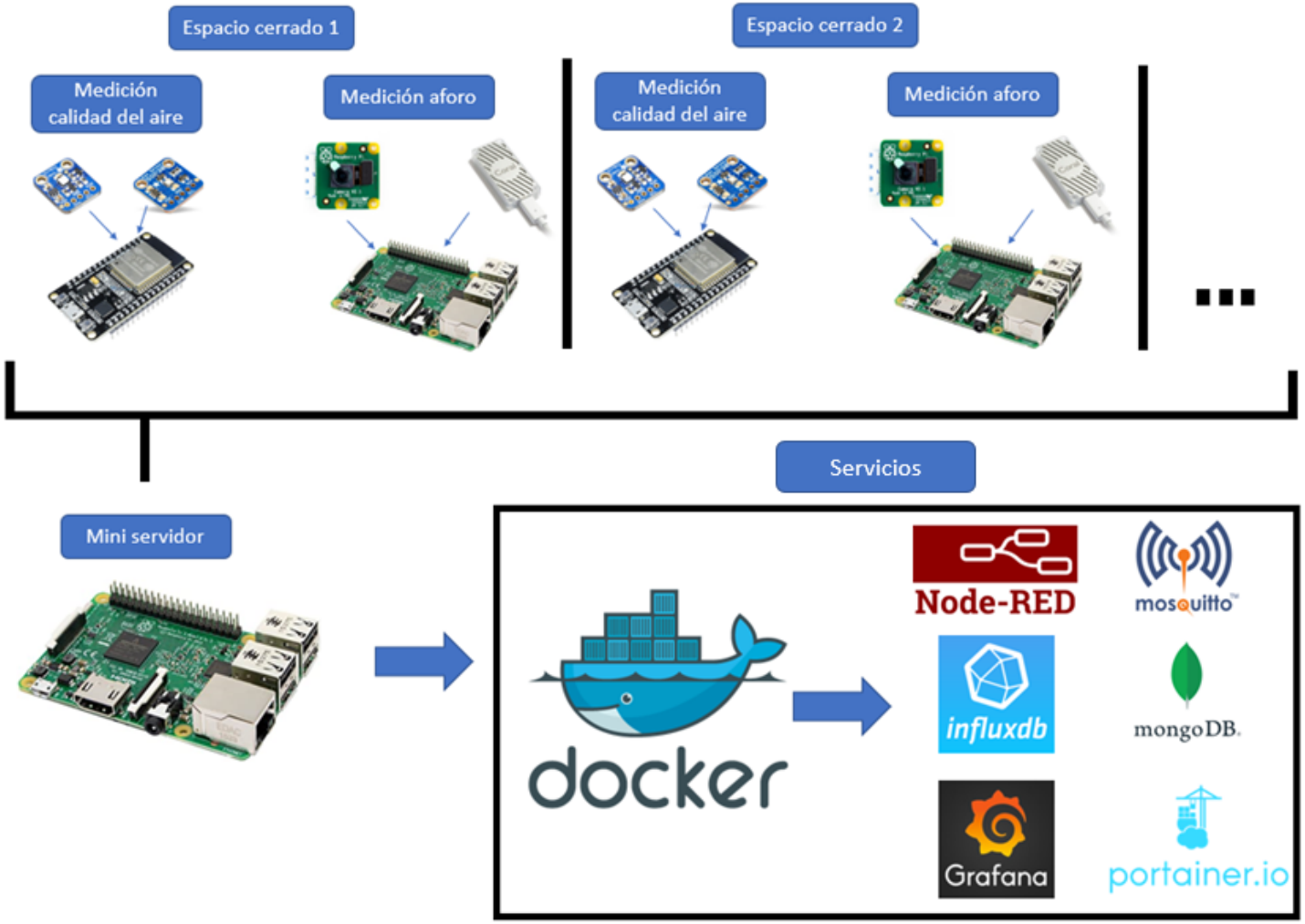


Figura 2.1 Esquema general del sistema implementado.

2.2. Medición de la calidad del aire

En este apartado se estudiará cómo implementar un componente capaz de medir la calidad del aire dentro de un espacio cerrado para saber si es necesario ventilar el espacio y reducir las probabilidades de contagiarse por COVID-19.

En la investigación sobre dispositivos capaces de medir atributos de la calidad del aire como el nivel de CO2, TVOC, temperatura o humedad, se encontraron los siguientes proyectos similares:

- **Phasak JD 3002**

  Phasak JD 3002 [1] es un dispositivo que ofrece datos sobre la calidad del aire, tales como, temperatura, humedad, CO2 y TVOC, permitiendo configurar los valores de alerta, para que emita avisos cuando se sobrepasen. Tiene un precio aproximado de unos 45 €, funcionando con una batería y una pantalla monocolor clásica.

Figura 2.2 Aspecto físico de Phasak JD 3002

Si este dispositivo se compara con nuestra solución de medición de calidad del aire, se podría decir que es demasiado grande y no tiene la capacidad de enviar los datos recopilados usando Wifi.

Así mismo Phasak JD 3002, cuenta con la capacidad de avisar si superan determinados valores, pero solo a los usuarios cercanos al dispositivo.

- Medidor de CO2 Sanico2 [2] es un dispositivo capaz de monitorizar la calidad del aire midiendo los niveles de CO2 y temperatura, pudiendo avisar si el espacio necesita ser ventilado para crear un espacio seguro frente al COVID-19. También cuenta con una aplicación para dispositivos móviles capaz de visualizar el estado del aire y de enviar alertas y notificaciones. Este dispositivo tiene un precio de 198 € por unidad.

  Si se compara con el sistema desarrollado, el medidor de CO2 Sanico2 se encuentra dentro de una caja que es capaz de variar de color en función de las mediciones que se toman, lo que lo hace vistoso para los usuarios e intuitivo. Los 3 colores que usa son: verde (niveles correctos), amarillo (niveles regulares) y rojo (niveles malos, airear el espacio).

  En el sistema desarrollado en este proyecto, no se hace necesario el desarrollo de una aplicación para dispositivos móviles, ya que se vale de la aplicación Telegram, la cual permitirá crear un bot que podrá ser programado para el envío de alertas.

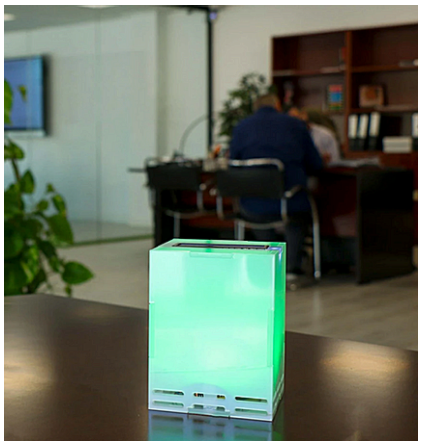


Figura 2.3 Aspecto físico del medidor de CO2 Sanico2

2.3. Medición del aforo

En este apartado se estudiará la posibilidad de usar un código capaz de realizar la inferencia de una red neuronal convolucional (CNN) pre entrenada. Está CNN tendrá

que ser capaz de detectar objetos dentro de una imagen. Concretamente, se trabajó con la idea de detectar personas dentro de un espacio cerrado, lo que ayudaría a respetar los límites de aforo para evitar sobrecargar el ambiente más rápido de lo previsto. Si los límites de aforo son respetados, se podrán reducir las probabilidades de contagio por COVID-19.

Durante la investigación de posibles códigos, resultó interesante el código que proporciona NVIDIA Jetson en su repositorio de GitHub [3], el cual está enfocado para ser utilizado en el dispositivo NVIDIA Jetson Nano(es parecido a una Raspberry Pi).

En este repositorio de GitHub se explica cómo hacer funcionar el código de detección de objetos usando Python. Este código permite ejecutar tanto modelos de detección de objetos pre entrenados como modelos propios, con el matiz de que no se usa ningún acelerador (dispositivo que mejora el rendimiento de la inferencia, es decir, que hace ejecutarse más rápido la CNN).

El problema que hace difícil trabajar con el código de NVIDIA Jetson y, por tanto su descarte para este proyecto, es que la inferencia se realiza en la librería de Python llamada "jetson", lo que complica adaptar el código para ser capaz de enviar la información vía MQTT con transporte SSL/TLS.

En cambio, el código de Google Coral [4], sí es capaz de trabajar con un acelerador, haciendo que la inferencia del modelo de detección de objetos sea más rápida. También, Google Coral hace más visible el código fuente de la inferencia, lo que permitió poder adaptarlo con mayor facilidad para tratar los objetos detectados y ver si son personas, para poder calcular el aforo aproximado en un espacio cerrado y enviar esta información vía MQTT con transporte SSL/TLS.

2.4. Monitorización de los datos

En este apartado se estudiarán diferentes herramientas que permitan dar una solución sencilla y elegante para poder procesar, almacenar y visualizar los datos recibidos vía MQTT con transporte SSL/TLS.

Para lograr este objetivo se investigaron herramientas que permitieran trabajar en la nube como pueden ser:

- **Microsoft Azure** [5], es un servicio de computación en la nube creado por Microsoft para construir, probar, desplegar y administrar servicios mediante el uso de sus centros de datos.

  Microsoft Azure resultó ser una opción interesante, pero se detectaron algunos problemas. Como la necesidad de pagar por el servicio y que realmente el usuario no tiene el control total de los datos, ya que se está usando la aplicación en la nube de Microsoft y no se sabe que puede estar haciendo con los datos.

- **ThingSpeak** [6], es un software de código abierto, que permite a los usuarios trabajar con dispositivos IoT.

  ThingSpeak es capaz de visualizar los datos recopilados, así como almacenarlos en la nube. Estos servicios pueden ser utilizados en su versión gratuita, aunque tiene límites, como un número máximo de canales (BBDD) que se pueden crear.

  Aun así, se sigue teniendo poco control de los datos y se acabó descartando, ya que los límites de la versión gratuita no eran suficientes para cubrir las necesidades futuras del proyecto.

Por otro lado, se investigaron herramientas con las que se pudiera trabajar de manera local y sin necesidad de pagar por su uso.

Para este proyecto se optó por utilizar Docker para gestionar los diferentes servicios que procesarán, almacenarán y visualizarán los datos, ya que daba mayor control al usuario sobre los datos monitorizados, evitando tener que pagar por usar los servicios. Estas herramientas serán explicadas con más detalle en el Capítulo 5.

3. Medición de la calidad del aire en un espacio cerrado

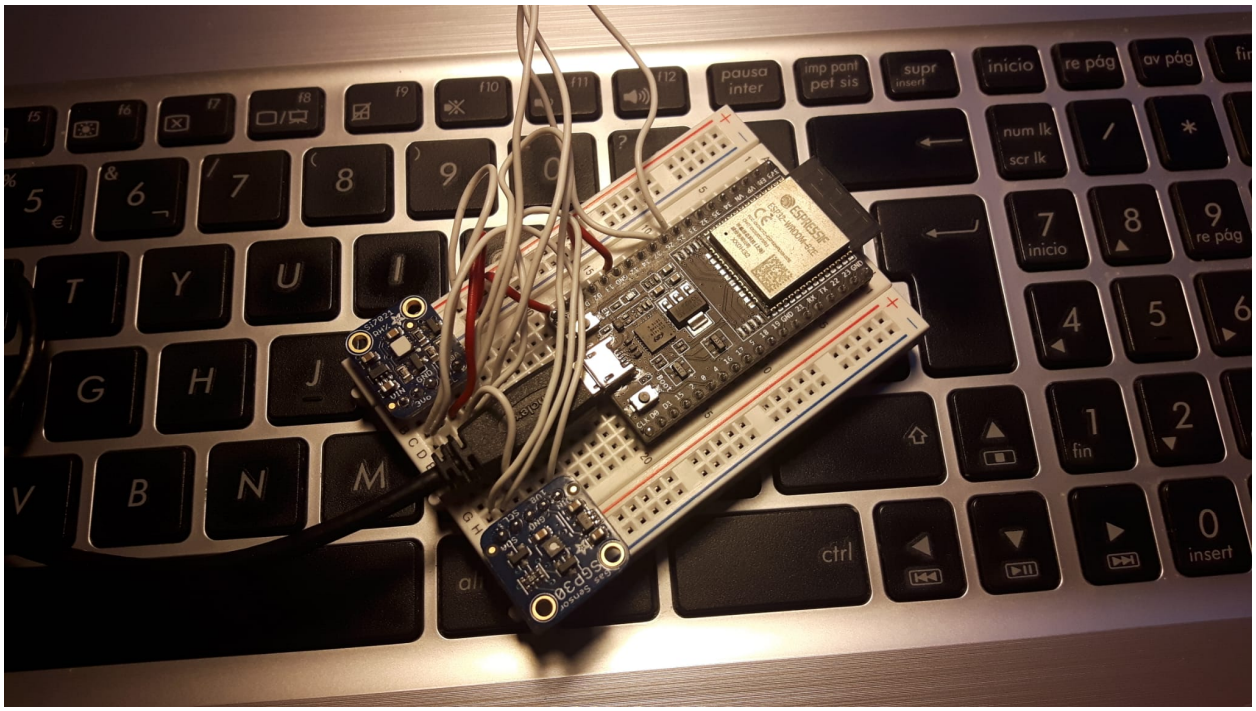


Figura 3.1 Aspecto físico del ESP32 conectado con los sensores SGP30 y Si7021

En este apartado se proporciona una solución a la manera de medir la calidad del aire en un espacio cerrado. Se investigó cómo afrontar la medición de la calidad del aire y qué dispositivos, tanto microcontroladores como sensores serían usados para realizar las mediciones, dando como resultado:

- **El microcontrolador ESP32**, ya que se tenía experiencia programando y que cubría las necesidades de poder conectarse con otros sensores y de enviar información vía Wifi.
- **Los sensores SGP30 y Si7021**, ya que se disponía de la documentación necesaria para saber cómo trabajar con ellos y que tenían la capacidad de medir los niveles de CO2, TVOC, temperatura y humedad en el ambiente.

Con todo esto, se implementó un programa software para que el ESP32 fuera capaz de recopilar y transmitir vía MQTT con transporte SSL/TLS, la información proporcionada por los sensores SGP30 y Si7021 a un mini servidor, desplegado en una Raspberry Pi.

3.1. Esquema general del nodo

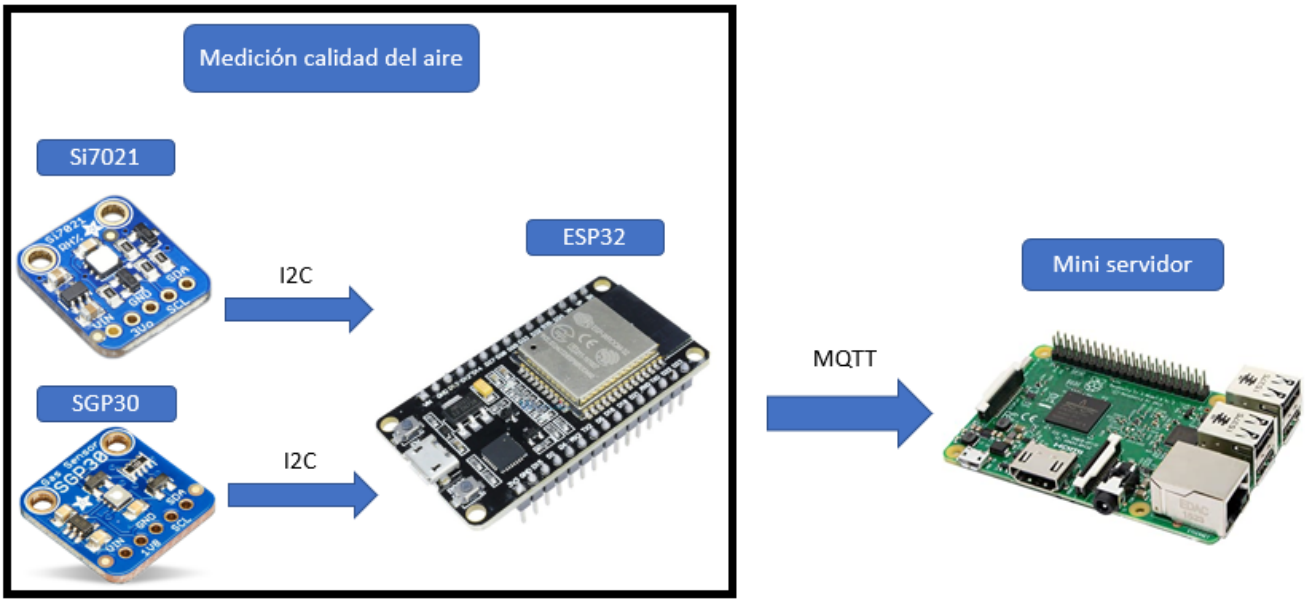


Figura 3.2 Esquema del nodo de medición de calidad del aire

Como se puede ver en la figura 3.2, el nodo desarrollado está formado por:

- **Un sensor Si7021**, encargado de tomar las mediciones de temperatura y humedad.
- **Un sensor SGP30**, encargado de tomar las mediciones de CO2 y TVOC.
- **Un microcontrolador ESP32**, encargado de comunicarse, usando I2C, con los sensores y de transmitir los datos recopilados de CO2, TVOC, temperatura y humedad vía MQTT con transporte SSL/TLS al mini servidor Raspberry Pi (este se encargará de monitorizar los datos enviados).

3.2. Flujo del código

A continuación se hablará, a alto nivel, del flujo de ejecución del código dentro del nodo encargado de la medición de la calidad del aire.

La ejecución empezará cuando el ESP32 sea conectado con el cable USB-microUSB a una fuente de alimentación.

Una vez tenga corriente, el ESP32 iniciará secuencialmente las siguientes tareas:

- Crea la conexión con la red Wifi.
- Arranca el módulo "**hora_fecha**":
    - Este módulo obtendrá la hora y fecha actual de un servidor SNTP, donde se indicará que tome la fecha correspondiente a su zona horaria. En el caso del proyecto, esa zona horaria será la de Madrid, España, indicado en la variable TZ_INFO, de la figura 3.3.

```
// Hora de Madrid, España
char* NTP_SERVER = "pool.ntp.org";//"es.pool.ntp.org";
char* TZ_INFO = "CET-1CEST-2,M3.5.0/02:00:00,M10.5.0/03:00:00";


static void obtain_time(){
    // Para sincronizar
    sntp_setoperatingmode(SNTP_OPMODE_POLL);
    sntp_setservername(0, NTP_SERVER);
    sntp_init();

    // wait for time to be set
    time_t now = 0;
    struct tm timeinfo = { 0 };
    int retry = 0;
    const int retry_count = 20;
    while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry < retry_count) {
        ESP_LOGI(TAG_HORA, "Waiting for system time to be set... (%d/%d)", retry, retry_count);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
    time(&now);
    localtime_r(&now, &timeinfo);
}
```

Figura 3.3 Función encargada de obtener la hora usando un servidor SNTP (variable NTP_SERVER)

    - Activará un "timer" periódico, que invocará a una función cada 15 segundos. Dentro de esta función se comprobará si es la hora de cambiar el modo de consumo energético.

```
// Declaro el timer periodico
esp_timer_handle_t periodic_timer3;
// Configuracion del timer periodico
const esp_timer_create_args_t periodic_timer_args2 = {
        .callback = &callback_timer,
        //.arg = (void*) medidas,
        .name = "periodic_timer_bajo_consumo"
};
// Creo el timer periodico
ESP_ERROR_CHECK(esp_timer_create(&periodic_timer_args2, &periodic_timer3));
// Arranco el timer periodico, cada hora se activara el timer
ESP_ERROR_CHECK(esp_timer_start_periodic(periodic_timer3, 15* 1000000));
```

Figura 3.4 Configuración de un "timer" periódico

Para cambiar el modo de consumo a "deep sleep" tendrán que ser más de las 22:00. Sino el ESP32 seguirá en modo "activo", como se puede ver en la figura 3.5.

```
static void callback_timer(void* arg) {
    int hora = get_hora_fecha();
    ESP_LOGW(TAG_HORA, "Hora: %d", hora);
    if(hora >= 22 || hora < 8){
        ESP_LOGW(TAG_HORA, "APAGAR EL SISTEMA");
        modo_deep_sleep();
    }
    else{
        ESP_LOGW(TAG_HORA, "SEGUIR");
    }

}
```

Figura 3.5 Función ejecutada por el "timer" periódico para decidir si se cambia el modo de consumo energético

- Arranca el módulo "**mqtts**"
    - Encargado de crear el cliente MQTT, que se usará para enviar la información de los sensores.
- Arranca el módulo "**sensores**"
    - Se llamará a una función encargada de realizar el muestreo de los valores utilizando I2C [7].

Este muestreo se llevará a cabo de la siguiente manera:

1. Se tomarán las mediciones de CO2 y TVOC del SGP30, donde se tomarán 5 muestras seguidas para calcular su media, quedándose con el valor de la media como medición. Después, de tomar 5 muestras de esta manera, se hará la media de estas 5 últimas muestras. El resultado de esta última medida será el valor que se guardará en una variable(tanto el valor de CO2 como el de TVOC).
2. Se tomarán las mediciones de temperatura y humedad del sensor Si7021 y se guardarán en una variable.

- Se activarán 2 "timers" periódicos:
  - El primer "timer" periódico invocará a una función cada segundo y servirá para publicar(enviar), vía MQTT con transporte SSL/TLS, los valores de los sensores, guardados en una variable de tipo "struct" que contendrá los 4 valores de los sensores (CO2, TVOC, temperatura y humedad).

    Antes de enviar los valores, estos se preparan en formato JSON, utilizando la librería "cJSON.h" [8].

```
// Funcion callback del timer periodico
static void callback_timer(void* arg) {
    /*if(empezar){
        tarea_publicar_sensores(co2, tvoc, temp, hum);
    }*/

    if(muestra.co2 == 0){
        valores.co2 = 400;
    }
    else{
        valores.co2 = muestra.co2;
    }

    valores.tvoc = muestra.tvoc;
    tarea_publicar_sensores((struct tValoresSensores *) &valores);

}
```

Figura 3.6 Función del primer timer periódico que llamara a la función que enviará los datos por MQTT

```c
// Tarea que publica los valores de los sensores con JSON
void tarea_publicar_sensores(void *args){
    tValoresSensores valores2 = *(tValoresSensores *) args;

    if(activo){
        printf("ACTIVO \n");

        cJSON *root = cJSON_CreateObject();
        //cJSON_AddStringToObject(root, "producto", "pulsera_ultra");
        cJSON_AddNumberToObject(root, "co2", valores2.co2);
        cJSON_AddNumberToObject(root, "tvoc", valores2.tvoc);
        cJSON_AddNumberToObject(root, "temperatura", valores2.temp);
        cJSON_AddNumberToObject(root, "humedad", valores2.hum);
        const char *json = cJSON_Print(root); //cJSON
        //printf("JSON: %s \n", json);

        //ESP_LOGE(TAG, "Valor topic: %s", topic_publi);FACULTAD_INFORMATICA/P_2/12/Sensor001/valores
        // QoS es el 2º empezando por la derecha
        int msg_id = esp_mqtt_client_publish(client_global, topic_publi_sensores, json, 0, 0, 0);

        free((void *)json);
        cJSON_Delete(root);
    }
    else{
        printf("DESACTIVADO \n");
    }
}
```

Figura 3.7 Función encargada de convertir en formato JSON y enviar la información vía MQTT

- El segundo "timer" periódico llamará a una función cada hora, este servirá para guardar las calibraciones del sensor de SGP30, ya que sino se hiciera, habría que esperar a que el sensor se calibre cada vez que se le corte la alimentación.

```c
// Funcion callback del timer periodico
static void callback_timer2(void* arg) {
    int ret;
    uint8_t sensor_data_h0, sensor_data_h1, sensor_data_h2, sensor_data_h3, sensor_data_h4, sensor_data_l;
    // get_baseline
    ret = i2c_master_sensor_sgp30(I2C_MASTER_NUM, &sensor_data_h0, &sensor_data_h1, &sensor_data_h2,
                                  &sensor_data_h3, &sensor_data_h4, &sensor_data_l, SGP30_GET_BASELINE);
    uint8_t lista[2];
    lista[0] = sensor_data_h0;
    lista[1] = sensor_data_h1;
    uint8_t crc_co2 = generateCRC(lista, 2);
    lista[0] = sensor_data_h3;
    lista[1] = sensor_data_h4;
    uint8_t crc_tvoc = generateCRC(lista, 2);
    if(crc_co2 == sensor_data_h2 && crc_tvoc == sensor_data_l){
        co2_baseline = (sensor_data_h0 << 8 | sensor_data_h1);
        tvoc_baseline = (sensor_data_h3 << 8 | sensor_data_h4);
        ESP_LOGI(TAG, "TODO BIEN");
        //ESP_LOGE(TAG, "base_line_co2: %d", co2_baseline);
        uint8_t primer = co2_baseline & 0xff;
        uint8_t segun = (co2_baseline >> 8);

        // Escribir el get_baseline
        write_nvs_partition("co2_baseline", co2_baseline);
        write_nvs_partition("tvoc_baseline", tvoc_baseline);
    }
    else{
        ESP_LOGI(TAG, "MAL AQUI");
    }
}
```

Figura 3.8 Función del segundo timer periódico para guardar las calibraciones del sensor SGP30.

- Arranca el módulo "**bajo_consumo**"
    - Se encargará de configurar el gestor de energía. El ESP32 se configuró de manera que si no tiene ninguna tarea ejecutándose, pasará automáticamente al modo "light sleep" [9]. Esta configuración se puede ver en la figura 3.9.

```c
// Power Save, cuando no tenga nada que hacer se metera en light sleep
esp_pm_config_esp32_t pm_config = {
    .max_freq_mhz = (int)240,// 240 MHz max
    .min_freq_mhz = (int)40,// 40 MHz min
    .light_sleep_enable = true
};
ESP_ERROR_CHECK(esp_pm_configure(&pm_config) );
```

Figura 3.9 Configuración del gestor de energía

    - En este módulo se declarará una función, que cuando se llame, cambiará el modo de consumo energético a "deep sleep" y un "timer"

será configurado para que cuando pasen 10 horas, despierte al ESP32 y vuelva al modo "activo", que es el modo por defecto. Esta función se puede observar en la figura 3.10.

```
// Funcion para cambiar el modo a deep sleep
void modo_deep_sleep(){
    // Configuro el timer que nos sacara del estado "deep sleep", si todo va bien
    const int wakeup_time_sec = 36000;//10 Estara en este estado durante 10 horas
    //printf("Enabling timer wakeup, %ds\n", wakeup_time_sec);
    ESP_ERROR_CHECK(esp_sleep_enable_timer_wakeup(wakeup_time_sec * 1000000));

    // Llamar al modo "deep sleep"
    //ESP_LOGI(TAG, "DESPERTADO");
    esp_deep_sleep_start();
}
```

Figura 3.10 Función que cambiará el modo de energía a "deep sleep"

Es necesario puntualizar:

- Los timers periódicos, se ejecutarán en todo momento, excepto cuando se pase a modo "deep sleep", en el cual permanecen desactivados.
- Cuando se despierte al ESP32 del modo "deep sleep", se volverá a ejecutar el flujo descrito, empezando otra vez por la creación de la conexión con la red Wifi, etc.

3.3. Microcontrolador ESP32

El ESP32 pertenece a la familia de microcontroladores desarrollados y distribuidos por la empresa china Espressif [10], la cual se encarga de poner a disposición de los desarrollados un framework oficial llamado "ESP-IDF"(**Esp**ressif **I**oT **D**evelopment **F**ramework) [11] y una serie de códigos de ejemplo en su cuenta oficial de GitHub [12] acompañados de documentación oficial [13]. Facilitando de esta manera, el desarrollo de software para este microcontrolador, incluso si surgiera algún tipo de duda a la hora de trabajar con el ESP32, Espressif da soporte a preguntas que pueda tener la comunidad mediante un foro oficial [14].

Figura 3.11 Visión general del ESP32

3.1.1. Características del ESP32

A continuación, se comentarán las características que posee el ESP32 con el que se trabajó en este proyecto:

- Procesador:
    - Microprocesador de 32-bits Tensilica Xtensa LX6 de doble núcleo, pudiendo operar en frecuencias de hasta 240 MHz
- Coprocesador de ultra bajo consumo o ULP (**u**ltra **l**ow **p**ower), permitiendo un ahorro de energía máximo cuando no se necesita el procesador principal.
- Modulo ESP32-WROOM-32D, el cual incluye:
    - Memoria flash de 4 Mb, donde se guardará el código a ejecutar y en el caso de que se sobrepase este tamaño, el microcontrolador se reiniciará constantemente.
    - Antena imprimida en la placa, la cual servirá para realizar comunicaciones inalámbricas vía Wifi y/o Bluetooth teniendo la capacidad de trabajar con BLE (**B**luetooth **L**ow **E**nergy), para el ahorro de energía en las comunicaciones.
- Placa de desarrollo ESP32-DEVKIT-C, la cual incluye:

- Regulador de tensión y controlador serie USB, para poder dar alimentación y programar el ESP32 desde el mismo conector micro USB.
- 36 pines de entrada/salida o GPIO, los cuales se describen en la figura 3.12.

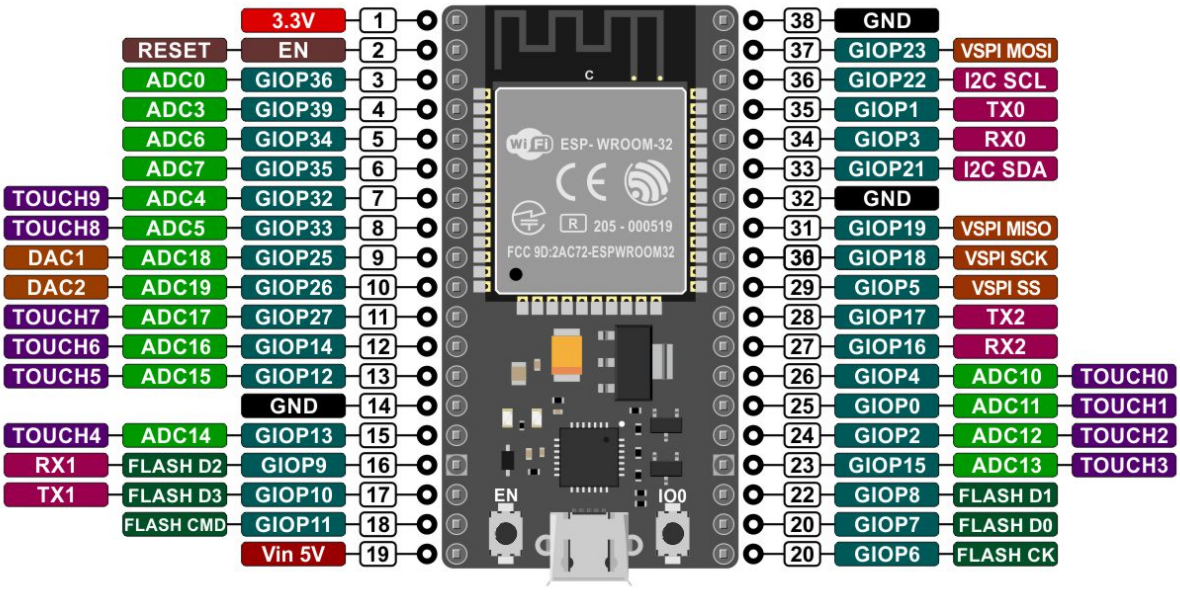


Figura 3.12 Descripción de los 36 pines disponibles en ESP32-DEVKIT-C

De todos ellos, se trabajará con cuatro:

- Pin 19, que será el que proporcione una energía de 5V para los sensores SGP30 y Si7021.
- Pin 14, que será la toma de tierra o GND para los sensores SGP30 y Si7021.
- Pin 10 o GPIO26, que se encargará de establecer la comunicación I2C del puerto SCL.
- Pin 9 o GPIO25, que se encargará de establecer la comunicación I2C del puerto SDA.

- Interfaces periféricas:
    - Sensor de efecto Hall, es un sensor incorporado en el ESP32, que permite detectar si hay un imán cerca y de esta manera utilizarlo para lo que se desee, como lanzar o parar una ejecución. [15]
    - I2C, el cual será utilizado para la comunicación entre el ESP32 y los sensores de medición de CO2, TVOC, temperatura y humedad. [7]
    - I2S, [16]
    - SPI, [17]
    - Bus CAN 2.0 [18]
    - Para saber más acerca de las interfaces periféricas del ESP32. [19] [20] [21]
- Bajo coste por unidad dependiendo de donde se compre, no superando los 10
- Gestión del consumo de energía, ofreciendo 5 modos de energía [22] :
    - **Modo activo**, es el modo por defecto de ESP32 y donde todas las características del chip están activas.

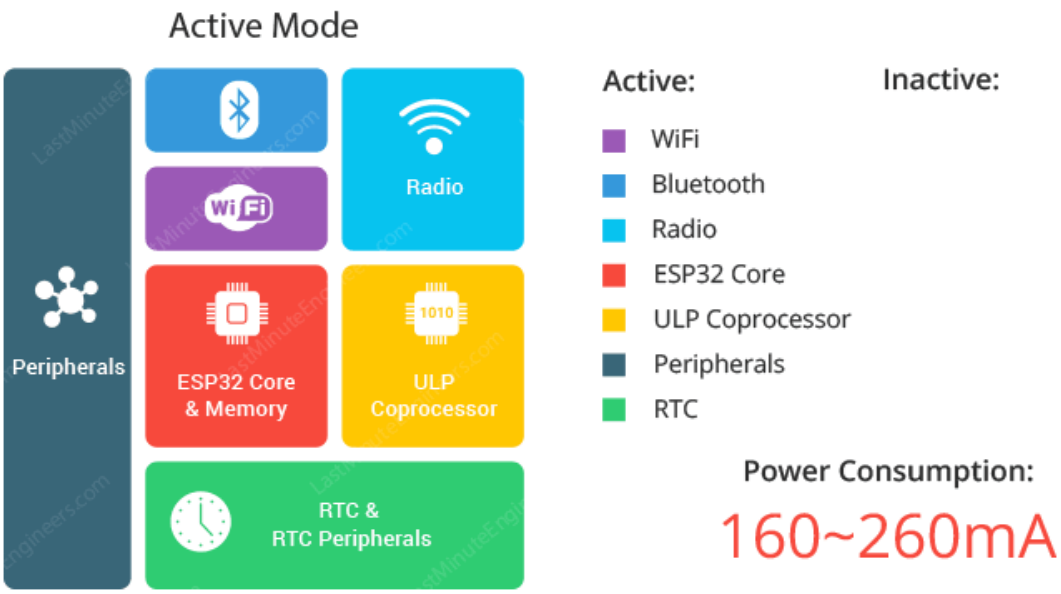


Figura 3.13 Resumen del modo "activo"

- **Modo de suspensión del modem**, en este modo quedan deshabilitados el Wifi, Bluetooth, radio y periféricos, aunque se puede configurar para despertar los módulos de Wifi y Bluetooth periódicamente.

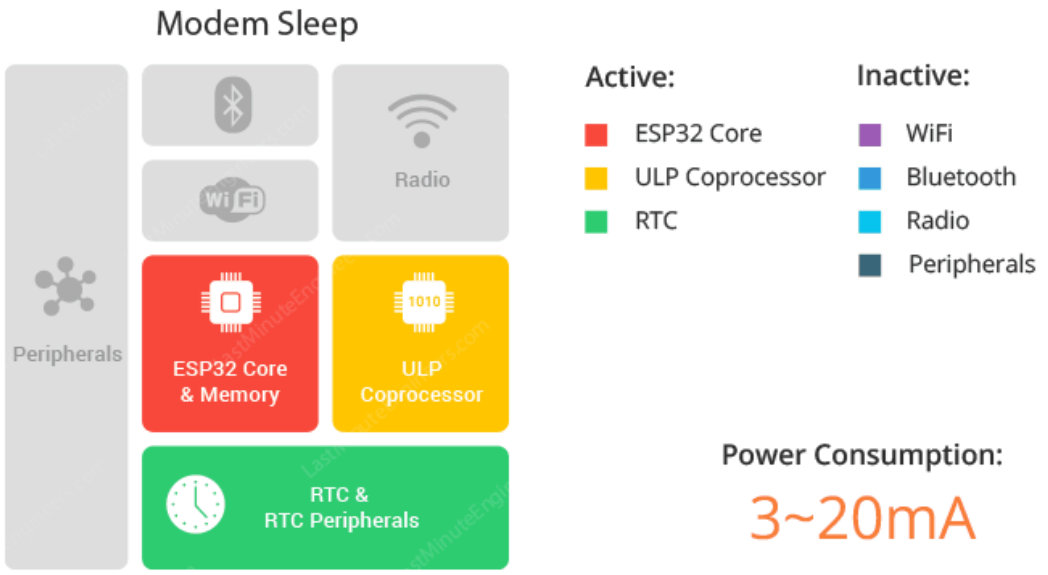


Figura 3.14 Resumen del modo "suspensión del modem"

- **Modo de sueño ligero o "light sleep"**, es parecido al modo de suspensión del modem, pero con el matiz de que el procesador del ESP32 se queda pausado, guardando un estado interno y se reactivará cuando salga de la suspensión.

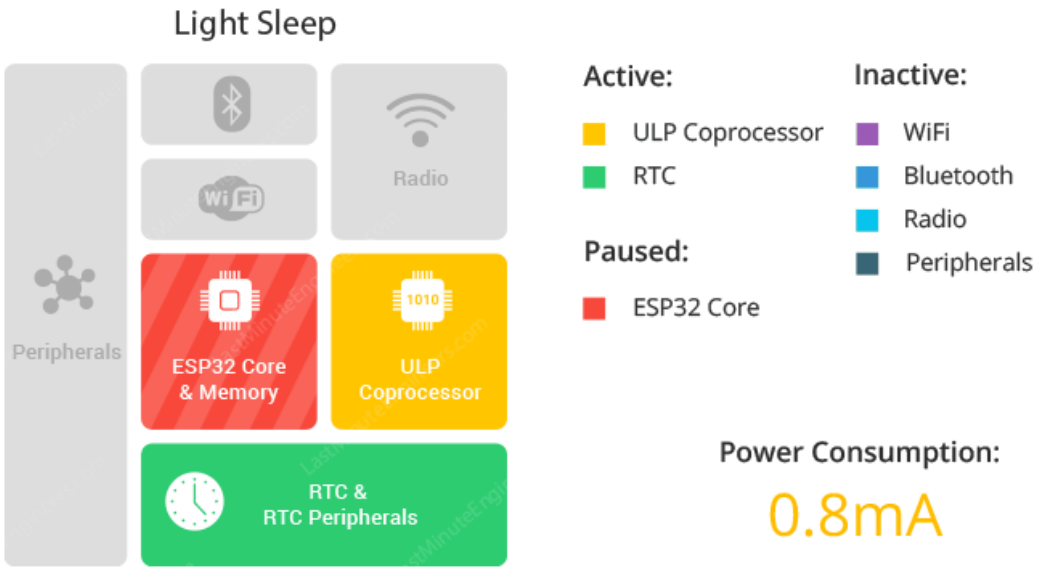


Figura 3.15 Resumen del modo "light sleep"

- **Modo de sueño profundo o "deep sleep"**, en este modo los módulos que se mantienen activos son el coprocesador ULP, RTC (temporizador, para salir del sueño una vez pase el tiempo programado) y los periféricos RTC.

  Este será el modo elegido para el proyecto, el cual se implementará para reducir el consumo del ESP32, cuando no sea necesario su uso, concretamente de 22:00 a 8:00 pasará a este modo.

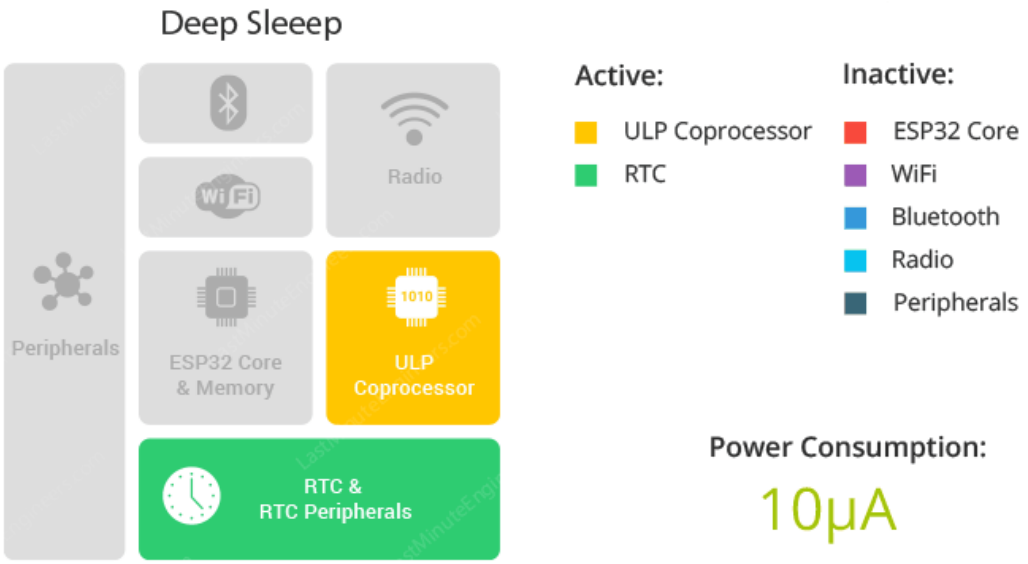


Figura 3.16 Resumen del modo "deep sleep"

- **Modo de hibernación**, este modo es el más radical para ahorrar en el consumo de energía, lo que implica tener solo activo el módulo RTC, junto a algunos GPIO RTC, que serán los que permitan salir de este modo.

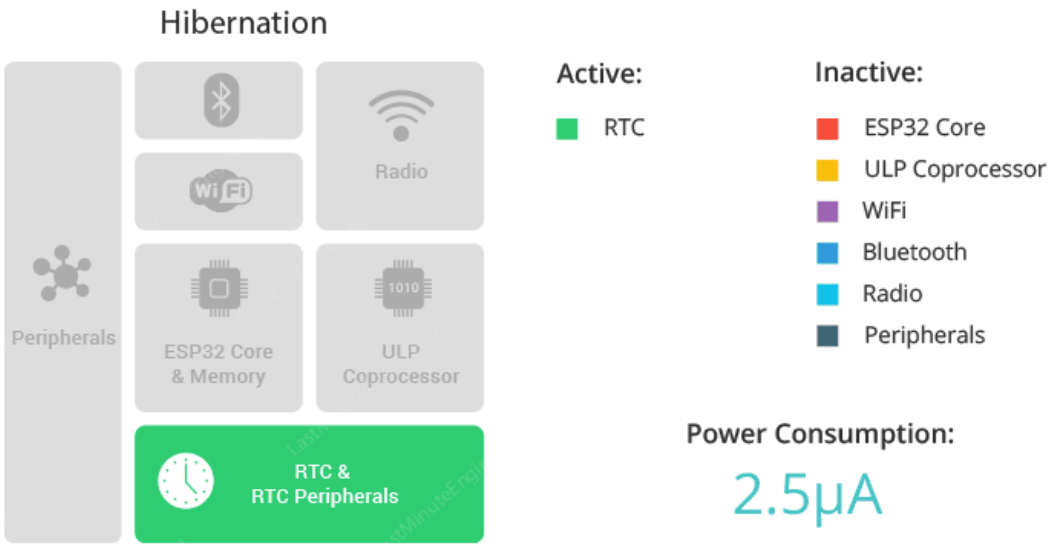


Figura 3.17 Resumen del modo "hibernación"

Gracias a los ejemplos de código que pone Espressif a disposición de los desarrolladores en su GitHub oficial [12], se pudieron adaptar las siguientes funcionalidades al ESP32:

- Comunicación vía MQTT con transporte SSL/TLS [23]
- Saber la hora a través de un servidor NTP, esto será de utilidad para que el ESP32 sepa qué hora es, pudiendo pasar al modo de "deep sleep" [24]
- Comunicación entre varios sensores a través de I2C [25]
- Uso de NVS (Non-Volatile Storage) para guardar los valores de calibración del sensor SGP30, evitando que se tenga que calibrar cada vez que se encienda el ESP32 [26]

3.1.2. Sensores utilizados para la medición de CO2, TVOC, temperatura y humedad

Para la tarea de medir los valores de CO2. TVOC, temperatura y humedad se realizó un estudio sobre diferentes sensores diseñados para tales fines y que se pudieran comunicar con el microcontrolador ESP32. Seleccionando finalmente los siguientes sensores:

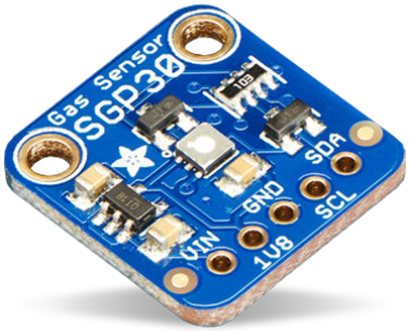


Figura 3.18 Sensor SGP30, para la medición del CO2 Y TVOC

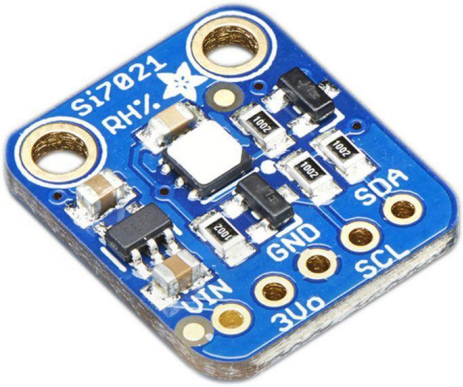


Figura 3.19 Sensor Si7021, para la medición de la temperatura y humedad

Ambos sensores poseen interfaz I2C, la cual permite manejar una comunicación segura entre dos dispositivos digitales, siendo uno el "maestro" (ESP32) y otros los esclavos (Si7021 y SGP30).

I2C es un puerto y protocolo de comunicación serial, define la trama de datos y las conexiones físicas para transferir bits entre dos dispositivos digitales. El puerto incluye dos cables de comunicación, SDA y SCL. Además, este protocolo permite conectar hasta 127 dispositivos esclavos a la vez.

Para conocer más en profundidad el funcionamiento de I2C. [27]

Para saber qué atributos usar para programar la comunicación I2C, es necesario echar un vistazo a la documentación de ambos sensores en sus respectivos "datasheets":

- Datasheet del sensor SGP30, [28]

- Datasheet del sensor Si7021, [29]

Donde se comentan los valores de los comandos a utilizar:

- Comandos para el sensor SGP30:

Measure Test

The command "Measure_test" which is included for integration and production line testing runs an on-chip self-test. In case of a successful self-test the sensor returns the fixed data pattern 0xD400 (with correct CRC).

Feature Set	0x0020				
Command	Hex. Code	Parameter length including CRC [bytes]	Response length including CRC [bytes]	Measurement duration [ms]	
				Typ.	Max.
Init_air_quality	0x2003	-	-	2	10
Measure_air_quality	0x2008	-	6	10	12
Get_baseline	0x2015	-	6	10	10
Set_baseline	0x201e	6	-	10	10
Set_humidity	0x2061	3		1	10
Measure_test[9]	0x2032	-	3	200	220
Get_feature_set_version	0x202f	-	3	1	2
Measure_raw_signals	0x2050	-	6	20	25

Figura 3.20 Tabla de comandos para leer las mediciones del sensor SGP30, página 9 del datasheet

Haciendo un breve resumen sobre los comandos:

- "**init_air_quality**", servirá para iniciar la medición de los niveles de CO2 y TVOC, esta operación necesitará 15 segundos para configurarse, tomando valores por defecto sobre las mediciones: 400 ppm para el CO2 y 0 ppb para TVOC, como aclaración TVOC es una medida adicional, la cual mide la concentración de compuestos orgánicos volátiles que se convierten fácilmente en vapores o gases, contribuyendo a la mala calidad del aire. [30]
- "**measure_air_quality**", servirá para recibir en el ESP32 las mediciones del sensor SGP30, las cuales vendrán juntas y se sacarán como se indica en el código adjunto a esta memoria, en el módulo "sensores.c"

- "**get_baseline**", servirá para leer los valores de calibración de las mediciones de CO2 y TVOC del SGP30, guardándolos en el ESP32 y poder recuperarlos cada vez que se apague y encienda el ESP32.
- "**set_baseline**", para modificar los valores de calibración de las mediciones de CO2 y TVOC del SGP30, cuando se encienda el ESP32.

- Comandos para el sensor Si7021:

Table 11. I²C Command Table

Command Description	Command Code
Measure Relative Humidity, Hold Master Mode	0xE5
Measure Relative Humidity, No Hold Master Mode	0xF5
Measure Temperature, Hold Master Mode	0xE3
Measure Temperature, No Hold Master Mode	0xF3
Read Temperature Value from Previous RH Measurement	0xE0
Reset	0xFE
Write RH/T User Register 1	0xE6
Read RH/T User Register 1	0xE7
Write Heater Control Register	0x51
Read Heater Control Register	0x11
Read Electronic ID 1st Byte	0xFA 0x0F
Read Electronic ID 2nd Byte	0xFC 0xC9
Read Firmware Revision	0x84 0xB8

Figura 3.21 Tabla de comandos para leer las mediciones del sensor Si7021, página 18 del datasheet

Más adelante se comenta la fórmula que se debe aplicar para obtener un valor coherente:

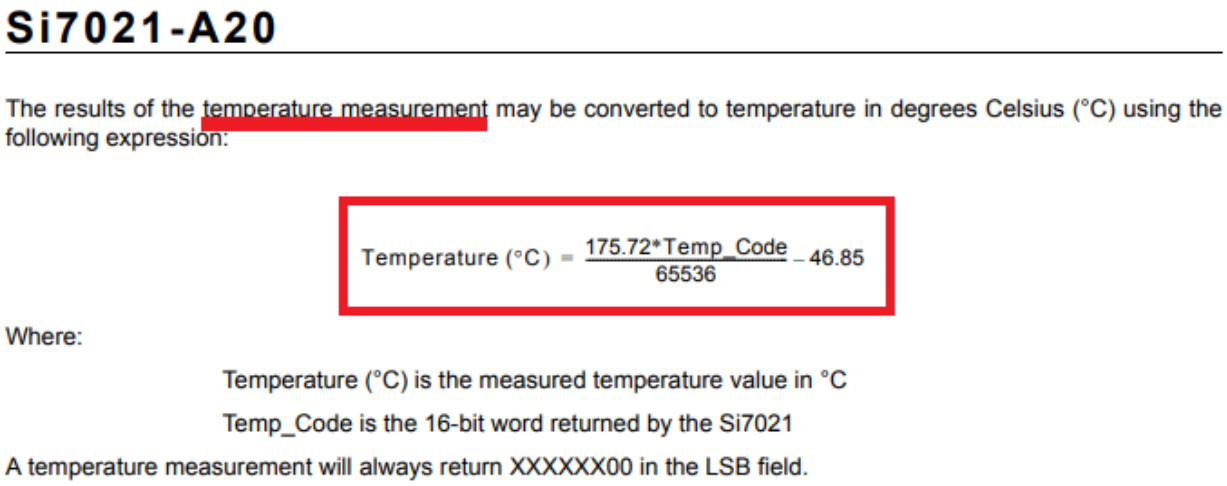
5.1.1. Measuring Relative Humidity

Once a relative humidity measurement has been made, the results of the measurement may be converted to percent relative humidity by using the following expression:

$$\%RH = \frac{125*RH\_Code}{65536} - 6$$

Where:

Figura 3.22 Fórmula para calcular la humedad detectada por el sensor Si7021, página 21 del datasheet



Si7021-A20

The results of the temperature measurement may be converted to temperature in degrees Celsius (°C) using the following expression:

$$\text{Temperature (°C)} = \frac{175.72 * \text{Temp\_Code}}{65536} - 46.85$$

Where:

Temperature (°C) is the measured temperature value in °C

Temp_Code is the 16-bit word returned by the Si7021

A temperature measurement will always return XXXXXX00 in the LSB field.

Figura 3.23 Fórmula para calcular la temperatura detectada por el sensor Si7021, página 22 del datasheet

Estos comandos serán usados para poder comunicarse con el sensor Si7021 y poder leer las mediciones de temperatura y humedad.

Para entender mejor su funcionamiento, se recomienda visualizar el código adjunto a la entrega de esta memoria, el apartado "ESP32", módulo "sensores.c".

3.4. MQTT

MQTT [31] fue el protocolo de comunicación elegido para transmitir por Wifi la información recogida tanto por los sensores Si7021 y SGP30 a través del microcontrolador ESP32, como por la cámara incorporada a una Raspberry Pi para la medición del aforo en un espacio cerrado. Recibiendo toda esta información, una segunda Raspberry Pi que hará las funciones tanto de bróker MQTT como de suscriptor a estos canales o topics, utilizando el transporte SSL/TLS, para una mayor seguridad y privacidad en las comunicaciones.

De acuerdo al autor del artículo [32], "MQTT son las siglas en inglés de **M**essage **Q**ueuing **T**elemetry **T**ransport; es un protocolo de comunicación M2M

(machine-to-machine) de tipo message queue [33] o cola de mensajes, una forma de comunicación asíncrona de servicio a servicio, que puede almacenar los mensajes en cola hasta ser procesados o eliminados."

MQTT fue creado por Andy Standford-Clark de IBM y Arlen Nipper de Arcom en 1999 como un mecanismo para conectar dispositivos empleados en la industria petrolera.

Inicialmente fue un formato propietario, en otras palabras, de uso privado y exclusivo, hasta 2010 donde fue liberado y pasó a ser un estándar en 2014 según OASIS (**O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards) [34].

El funcionamiento del MQTT es un servicio de mensajería de publicación/suscripción, donde los clientes se conectan a un servidor central denominado bróker.

Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en "topics" organizados jerárquicamente; de esta manera un cliente puede publicar un mensaje en un determinado topic, lo que permite que otros clientes pueden suscribirse a este topic y el bróker le hará llegar los mensajes a los suscriptores del topic.

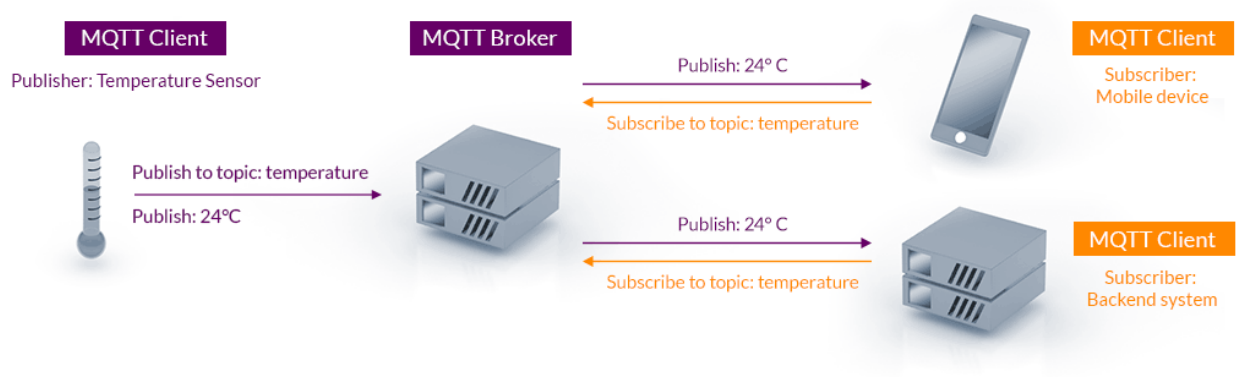


Figura 3.24 Arquitectura de publicación / suscripción de MQTT

Los clientes inician una conexión TCP/IP con el bróker, el cual mantendrá un registro de los clientes conectados y mantendrá esta conexión abierta hasta que el cliente la finalice.

Por defecto, MQTT emplea los puertos:

- 1883, este puerto es usado para enviar mensajes sin encriptación
- 8883, este puerto es usado para enviar mensajes con encriptación TLS

Ahora se pasará a citar sus puntos fuertes que lo hacen ideal para el uso en dispositivos IoT:

- **Ligereza y eficiencia**, ya que los clientes MQTT son muy pequeños y, por tanto, requieren recursos mínimos, pudiendo ser utilizados en dispositivos de escasa frecuencia como el ESP32. También, permite optimizar el ancho de banda de la red, gracias a los mensajes de encabezados pequeños, lo que se traduce esta menor necesidad de recursos, en un menor consumo energético.
- **Permite una comunicación bidireccional**, facilitando tanto enviar mensajes como recibirlos en una comunicación entre dispositivos y la nube y viceversa.
- **La capacidad de escalar**, permitiendo conectarse con millones de dispositivos IoT.
- **Entrega de mensajes confiable**, teniendo 3 niveles de calidad del servicio, también conocido como QoS:
    - **QoS 0 (at most one)**, el mensaje se envía una única vez. En caso de haber un fallo, puede que algún mensaje no se entregue. Esta opción será utilizada en este proyecto en la parte de envío de mediciones de CO2, TVOC, temperatura y humedad, ya que el envío de estas mediciones es cada segundo y realmente no importa si se pierde algún mensaje.
    - **QoS 1 (at least one)**, el mensaje se envía hasta que se garantiza la entrega. En caso de fallo, el suscriptor puede recibir algún mensaje duplicado.
    - **QoS 2 (exactly one)**, se garantiza que el mensaje se entrega al suscriptor, y únicamente una vez. Esta opción será utilizada en este proyecto en la parte de detección del aforo.
- **Seguridad habilitada**, disponiendo de distintas medidas de seguridad, entre las cuales se incluyen:
    - Transporte SSL/TLS
    - Autenticación por usuario y contraseña
    - Mediante certificados

Aunque el uso de mecanismos de seguridad se hace bastante interesante y en algunos casos necesario, cabe recordar que conlleva una carga de proceso importante para dispositivos IoT con escasa capacidad.

3.4.1. Generador de certificados

En el proyecto se optará por la opción de MQTT con transporte SSL/TLS, empleando para ello un certificado. Este certificado se generará con el script de la figura 3.25, que se encontrará en la carpeta de "script adicionales" y fue extraído de este GitHub [35] .

Se usará un script en bash para la generación automática de certificados usando SSL(Secure Sockets Layer), como se ve en la figura 3.25. El certificado con nombre "ca.crt" se usará para el envío y la recepción de mensajes vía MQTT con transporte SSL/TLS. El único ajuste que se deberá hacer es poner la IP del bróker MQTT donde se haya instalado, en el caso del proyecto será la IP del mini servidor en una Raspberry Pi [36].

```
#!/bin/bash

IP="192.168.1.81"
SUBJECT_CA="/C=SE/ST=Stockholm/L=Stockholm/O=himinds/OU=CA/CN=$IP"
SUBJECT_SERVER="/C=SE/ST=Stockholm/L=Stockholm/O=himinds/OU=Server/CN=$IP"
SUBJECT_CLIENT="/C=SE/ST=Stockholm/L=Stockholm/O=himinds/OU=Client/CN=$IP"

function generate_CA () {
   echo "$SUBJECT_CA"
   openssl req -x509 -nodes -sha256 -newkey rsa:2048 -subj "$SUBJECT_CA"  -days 365 -keyout ca.key -out ca.crt
}

function generate_server () {
   echo "$SUBJECT_SERVER"
   openssl req -nodes -sha256 -new -subj "$SUBJECT_SERVER" -keyout server.key -out server.csr
   openssl x509 -req -sha256 -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365
}

function generate_client () {
   echo "$SUBJECT_CLIENT"
   openssl req -new -nodes -sha256 -subj "$SUBJECT_CLIENT" -out client.csr -keyout client.key
   openssl x509 -req -sha256 -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 365
}

function copy_keys_to_broker () {
   sudo cp ca.crt /home/pi/Desktop/docker/mosquitto/config/certs/
   sudo cp server.crt /home/pi/Desktop/docker/mosquitto/config/certs/
   sudo cp server.key /home/pi/Desktop/docker/mosquitto/config/certs/
}

generate_CA
generate_server
generate_client
copy_keys_to_broker
```

Figura 3.25 Aspecto del generador de certificados usando SSL

3.4.2. Formato JSON

El formato utilizado en los mensajes MQTT será JSON.

De acuerdo a Wiki : JSON (Javascript Object Notation) [37] es un formato de texto sencillo para el intercambio de datos, basado en los objetos de Javascript. JSON se recomienda utilizar en entornos donde el tamaño del flujo de datos es de vital importancia.

JSON fue muy importante en el proyecto, ya que sirvió para enviar información entre los distintos nodos sensores, y para el transporte de los datos por la aplicación Node Red entre sus distintos nodos.

3.5. Configuración del nodo (ESP-IDF)

En este apartado se hablará como configurar el ESP32 utilizando el framework ESP-IDF.

3.5.1. ESP-IDF (Espressif IoT Development Framework)

Todo el desarrollo del código para el lado del ESP32 será posible utilizando el framework oficial de Espressif "ESP-IDF", el cual puede ser instalado en cualquier sistema operativo siguiendo los pasos del tutorial [38] y gracias a este framework se pudo trabajar en el desarrollo de software para el ESP32 usando los siguientes comandos:

- Comando "**idf.py menuconfig**", este sirve para acceder al menú de configuración del ESP32, como se puede ver en la figura 3.26.

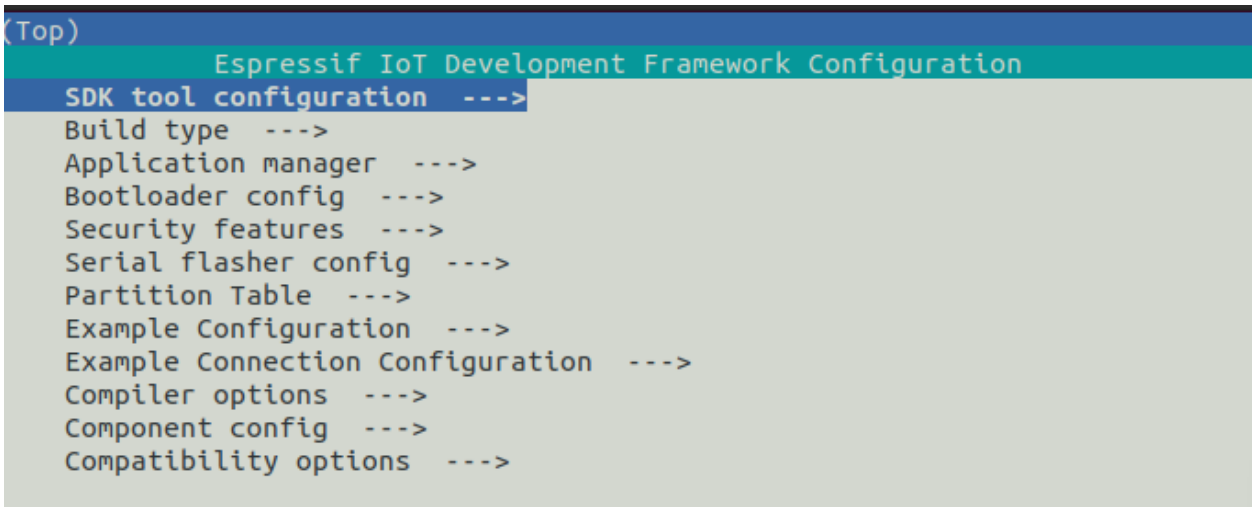


Figura 3.26 Aspecto del menú de configuracion para el ESP32

Ahora se pasará a hablar de las configuraciones disponibles para el ESP32, si se desea volver hacia atrás en algún menú pulsar la tecla "ESC":

  - Para configurar las credenciales de la red Wifi, desde el menú principal mostrado en la figura 3.26. Seguir las figuras 3.27 y 3.28

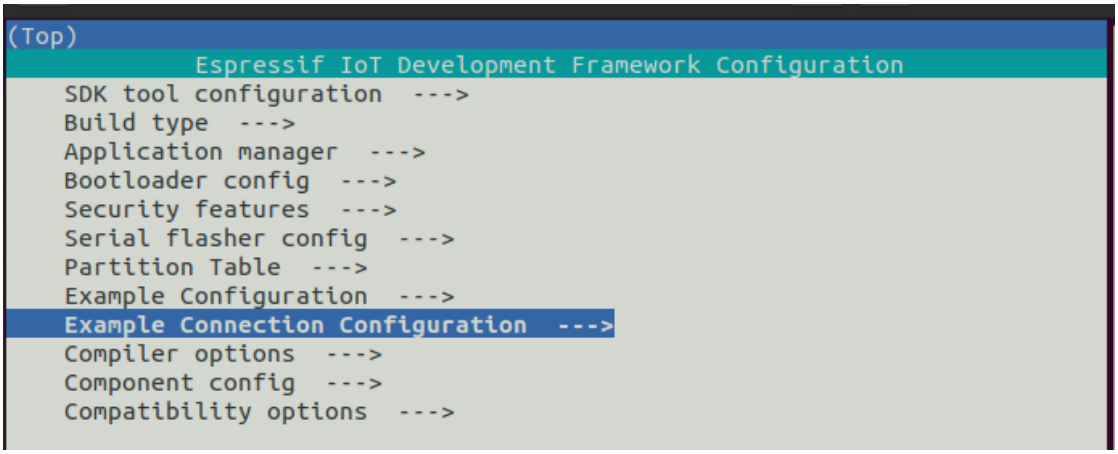


Figura 3.27 Primer paso para la configuración de las credenciales

Figura 3.28 Segundo paso para la configuración de las credenciales

Los campos "WiFi SSID" y "WiFi Password" se rellenarán con las credenciales de la red Wifi a la que se quiera conectar el ESP32.

- Para configurar la URL del bróker de MQTT, seguir la figura 3.29.

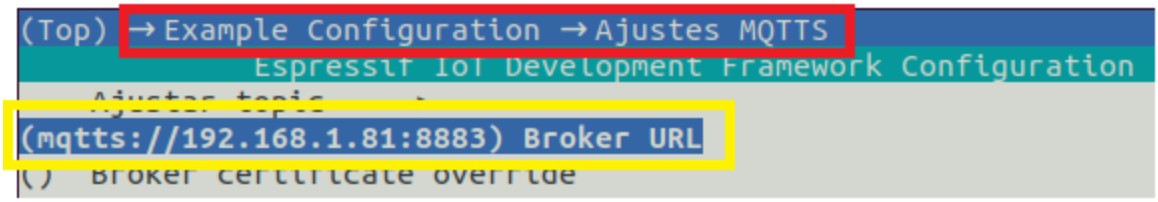


Figura 3.29 Cuadrado rojo, ruta desde el menú principal; cuadrado amarillo ajustes para la URL del bróker MQTT

- Para configurar el TOPIC de MQTT, siguiendo la figura 3.30, los TOPICs sirven para identificar a los sensores.

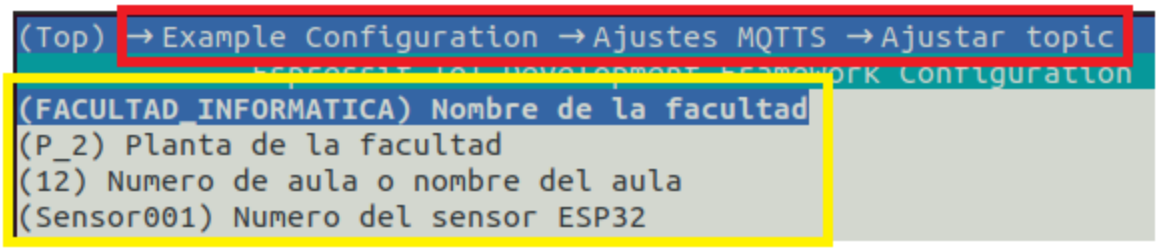


Figura 3.30 Cuadrado rojo, ruta desde el menú principal; cuadrado amarillo ajustes del TOPIC

- Para ajustar los pines que conectan con los sensores SGP30 y Si7021, usando el protocolo I2C, seguir la figura 3.31. Estos pines serán los usados para leer los datos de las mediciones de los sensores Si7021 y SGP30, concretamente los pines SDA y SCL, que estarán conectados al ESP32.

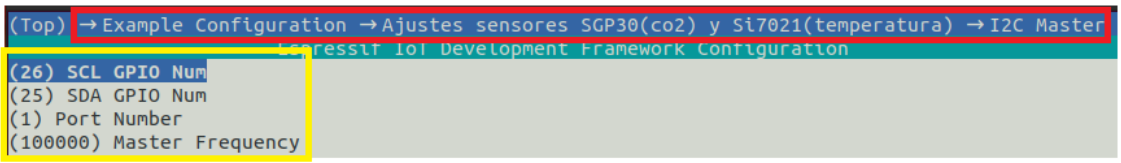


Figura 3.31 Cuadrado rojo, ruta desde el menú principal; cuadrado amarillo ajustes de los pines para el protocolo I2C

- Para configurar las variables de lectura de los sensores, seguir la figura 3.32

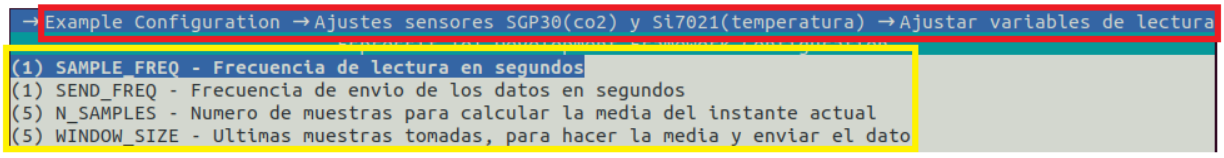


Figura 3.32 Cuadrado rojo, ruta desde el menú principal; cuadrado amarillo ajustes de las variables de lectura de los sensores

Pudiendo modificar este menú de configuración en el fichero llamado "Kconfig.projbuild", para conocer más acerca sobre el lenguaje empleado. [39]

Para los siguientes comandos, será necesario que el ESP32 esté conectado al ordenador por USB.

- Comando "**idf.py --port /dev/ttyUSB0 flash**", este servirá para compilar y flashear el código al ESP32, indicando en la opción de "--port" el puerto donde está conectado el ESP32, ya que en este caso se usó una máquina virtual de Ubuntu, la ruta para acceder a este puerto será "/dev/ttyUSB0".

  Si saltase algún error en la compilación, el "ESP-IDF", pararía y mostraría el mensaje de error, ya bien por fallo en la configuración del "menuconfig" o por un fallo en el propio código que se está intentando cargar al ESP32.

- Comando "**idf.py --port /dev/ttyUSB0 monitor**", este comando es similar al comando "flash", con el matiz de que se sustituye la palabra "flash" por "monitor" y se deberá ejecutar una vez se haya cargado el código al ESP32. Una vez ejecutado en la consola de comandos, esta mostrará mensajes informativos acerca de la ejecución del código, pudiendo ver si el código funciona según lo esperado o por el contrario requiere de alguna acción por parte del desarrollador.

4. Medición del aforo en un espacio cerrado

Figura 4.1 Aspecto físico del Raspberry Pi con cámara y acelerador USB Coral

En este apartado se investigaron diferentes dispositivos para la medición del aforo en un espacio cerrado, donde se decidió finalmente trabajar con una Raspberry Pi, la cual tendrá la "Cámara v2.1" incorporada para suministrar imágenes y estas serán tratadas con el código que pone a disposición Google Coral [4], el cual sirve para trabajar con modelos de detección de objetos.

En concreto, Google Coral aporta modelos de detección de objetos pre entrenados y listos tanto para trabajar con o sin el Edge TPU de Coral. Estos modelos serán capaces de detectar hasta 90 objetos diferentes, incluyendo personas, lo que ayudará en el proyecto a la estimación aproximada del aforo de un espacio cerrado.

4.1. Esquema general del nodo

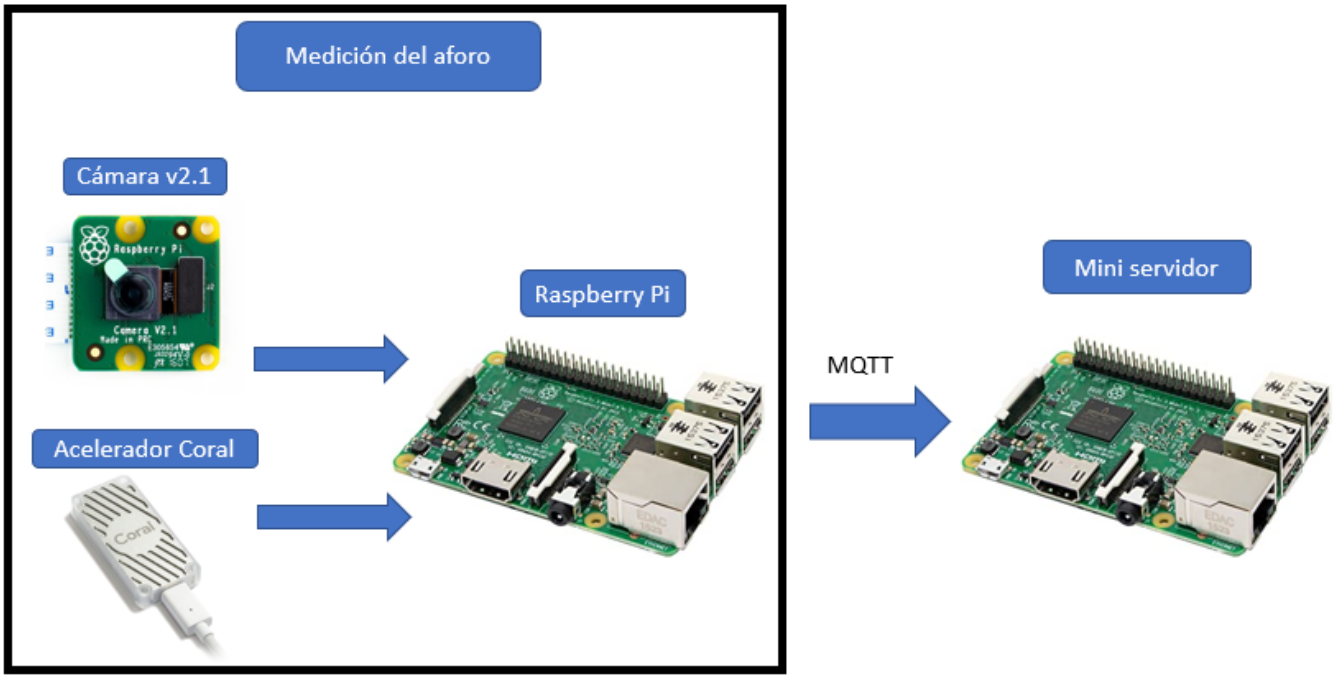


Figura 4.2 Esquema del nodo de medición del aforo

Como se puede ver en la figura 4.2, el nodo desarrollado está formado por:

- **Una cámara**, encargada de suministrar imágenes al código "adaptado" de Google Coral para procesarlas.
- **Un acelerador Coral**, encargado de mejorar la inferencia del modelo de detección de objetos.

- **Una Raspberry Pi**, encargada de ejecutar el código "adaptado" de Google Coral, el cual cargará el modelo de detección de objetos y realizará la inferencia. El número de objetos detectados y clasificados como "personas", será enviado vía MQTT con transporte SSL/TLS al mini servidor Raspberry Pi (este se encargará de monitorizar los datos enviados).

4.2. Flujo del código

La ejecución empezará cuando la Raspberry Pi sea conectada con el cable microUSB a una fuente de alimentación.

Una vez tenga corriente, la Raspberry Pi arrancará e iniciará secuencialmente las siguientes tareas:

- Se ejecuta un fichero .bash, que fue configurado en "sudo crontab -e"

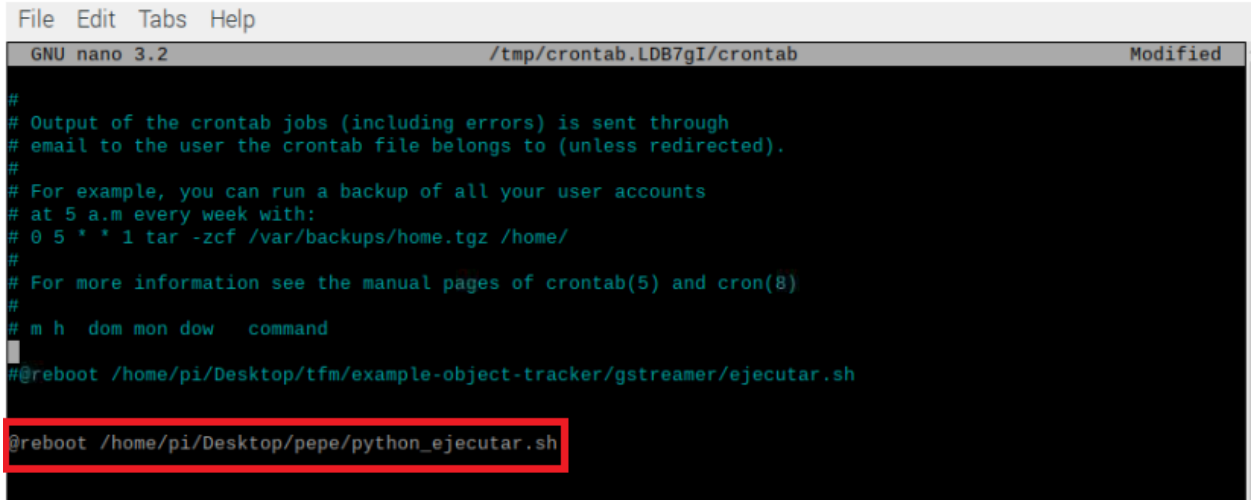


Figura 4.3 Configuración del sudo "crontab -e"

Este fichero .bash esperara 35 segundos, este intervalo de tiempo se hace necesario para dar tiempo a la Raspberry Pi a arrancar y después de esta pausa se ejecutará un segundo fichero .bash.

- Este segundo fichero .bash , se encargará de ejecutar el código "adaptado" de Google Coral. Dentro de este segundo fichero .bash se llamará al fichero python encargado de ejecutar el código y se incluirá un parámetro llamado "threshold", el cual permite indicar al programa que solo trate los objetos

detectados con una puntuación superior a la indicada en esta variable. En el caso del proyecto, el threshold será del 50%, como se puede ver en la figura 4.4.

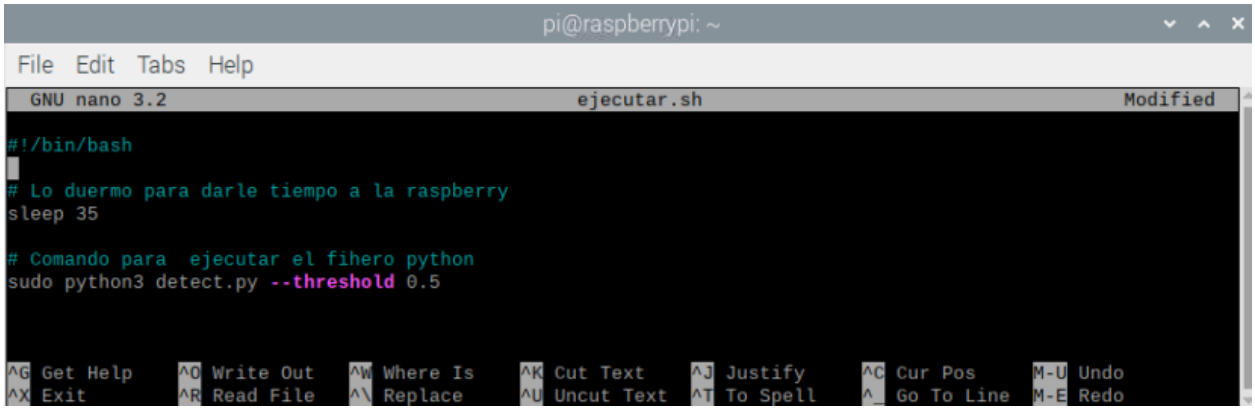


Figura 4.4 Configuración del segundo fichero .bash

- En la ejecución del código:
  - Se cargará el modelo de detección de objetos "SSD-MobileNet v2"
  - Se creará un cliente MQTT, para poder enviar la información recopilada, como se muestra en la figura 4.5. Para su implementación se utilizó la librería de python "paho-mqtt" [40].

```
def arrancar_mqtt():
    #define callbacks
    def on_message(client, userdata, message):
        print("Recibido")
        #print("received message =",str(message.payload.decode("utf-8")))

    """ def on_log(client, userdata, level, buf):
        print("log: ",buf) """

    def on_connect(client, userdata, flags, rc):
        print("publishing ")
        #client.publish("muthu","muthupavithran",)


    client=paho.Client()
    client.on_message=on_message
    #client.on_log=on_log
    client.on_connect=on_connect
    print("connecting to broker")
    client.tls_set("./ca.crt", tls_version=ssl.PROTOCOL_TLSv1_2)
    client.tls_insecure_set(True)
    client.connect("192.168.1.81", 8883, 60)

    ##start loop to process received messages
    client.loop_start()
    return client
```

Figura 4.5 Función que crea el cliente MQTT

- A partir de aquí se ejecuta en bucle:
  - Se realizará la inferencia del video en tiempo real, proporcionado por la "cámara v2.1". La inferencia será más rápida gracias al acelerador Coral conectado a la Raspberry Pi.
  - Después de la inferencia, se revisará que objetos detectados pertenecen a la clase "personas".

```python
def user_callback(input_tensor, src_size, inference_box, mot_tracker, num_personas, ids_dic, client_mqtt, tiempo_inicio, intervalo_mqtt):
    nonlocal fps_counter
    start_time = time.monotonic()
    common.set_input(interpreter, input_tensor)
    interpreter.invoke()
    # For larger input image sizes, use the edgetpu.classification.engine for better performance
    objs = get_output(interpreter, args.threshold, args.top_k)
    end_time = time.monotonic()
    detections = []  # np.array([])

    contador = 0

    tiempo_final = time.monotonic()
    for n in range(0, len(objs)):
        if objs[n].id == 0:# NUEVO, cogera solo personas
            element = []  # np.array([])
            element.append(objs[n].bbox.xmin)
            element.append(objs[n].bbox.ymin)
            element.append(objs[n].bbox.xmax)
            element.append(objs[n].bbox.ymax)
            element.append(objs[n].score)  # print('element= ',element)
            detections.append(element)  # print('dets: ',dets)
    # convert to numpy array #      print('npdets: ',dets)
    detections = np.array(detections)
    #print("Total personas en la imagen: ", len(detections))
    # Compruebo si hay que enviar
    if (tiempo_final - tiempo_inicio) >= intervalo_mqtt:
        #tiempo_inicio = time.time()
        tiempo_inicio = time.monotonic()
        enviar_MQTT(len(detections), client_mqtt)
```

Figura 4.6 Función donde se hace la inferencia del modelo, se filtran los objetos detectados como "personas", ya que estos pertenecen a la clase "0" y cada minuto se invoca a la función "enviar_MQTT()"

- Por último, cada minuto se enviará el aforo detectado vía MQTT con transporte SSL/TLS.

```python
def enviar_MQTT(contador, client_mqtt):
    topic = 'detecion_camara/1'
    msg='{ "aforo":' + str(contador) + '}'

    # Publica la info cifrada
    client_mqtt.publish(topic, msg, qos=2)

    print("Envio")
```

Figura 4.7 Función que envía el aforo vía MQTT

4.3. Visión por computador en Raspberry Pi

4.3.1. MobileNet y Single-Shot Detector (SSD)

En el proyecto se hará uso del modelo SSD-MobileNet v2, estando ya pre entrenado por Google Coral usando el conjunto de datos COCO17 [41] .

El modelo SSD-MobileNet v2 fue elegido para el proyecto, ya que se basa en una solución eficiente compuesto en dos fases:

- **Fase de extracción de características**, en base a una red neuronal genérica como es MobileNet, usando la versión 2.
- **Fase de detección de objetos**, en base a una segunda red neuronal específica como es SSD.

El aprendizaje profundo o "deep learning" ha impulsado un enorme progreso en el campo de la visión por computador en los últimos años, con redes neuronales cada vez más complejas y eficientes, capaces de detectar objetos con mayor precisión.

Para llevar a cabo la tarea de ejecutar de manera rápida y con alta precisión en un entorno con recursos limitados y optimizando el consumo de energía, Google AI desarrolló el modelo MobileNet, siendo este una familia de modelos de visión por computador enfocado en dispositivos móviles para Tensor Flow. Teniendo como características, que son modelos pequeños, de baja latencia y bajo consumo energético, parametrizados para cumplir con las limitaciones de recursos. MobileNet se podrá usar para clasificación, detección y segmentación de forma similar a como se utilizan otros modelos populares a gran escala como Inception [42] [43].

Para entender en profundidad cómo funciona SSD-MobileNet v2 se recomienda ver los siguientes artículos. [44] [45] [46] [47] [48] [49] [50] [51] [52]

4.3.2. Edge TPU Coral o acelerador USB Coral

El acelerador USB Coral [53] proporciona un coprocesador Edge TPU, permitiendo tener una alta velocidad en la inferencia de aprendizaje automático en una amplia gama de sistemas, conectándolo a un puerto USB.

El coprocesador Edge TPU es capaz de realizar 4000 millones de TOPS (tera operaciones por segundo), utilizando 0,5 vatios por cada TOPS. Pudiendo ejecutar modelos de visión por computador de última generación, como MobileNet V2 a casi 400 FPS(frames por segundo), de manera energéticamente eficiente.

Soporta cualquier sistema que ejecute Debian Linux (incluyendo Raspberry Pi), macOS o Windows 10, siendo compatible con TensorFlow Lite [54], lo que significa que no hace

falta construir modelos desde cero, ya que usando los modelos pre entrenados por Tensor Flow Lite se pueden compilar para ser ejecutados en Edge TPU.

El acelerador Coral, será utilizado en este proyecto para poder realizar las operaciones de inferencia del modelo de detección de objetos SSD-MobileNet v2, de manera más rápida.

4.3.3. Cámara Raspberry Pi - "Camera V2.1"

La cámara V2.1 [55] tiene un sensor Sony IMX219 [56] de 8 megapíxeles, siendo capaz de tomar videos de alta definición, admitiendo modos de vídeo de 1080p30 y 720p60. También puede usarse para tomar imágenes fijas.

En este proyecto servirá para que el modelo SSD-MobileNet v2 pueda detectar las personas de una imagen para poder calcular, de forma aproximada, el aforo en un espacio cerrado.

5. Monitorización de los datos generados

Figura 5.1 Aspecto físico del mini servidor Raspberry Pi

En este apartado se implementó un sistema capaz de procesar, almacenar y visualizar los datos recogidos y enviados por los diferentes sensores, de una manera sencilla, donde se trabajó con la herramienta Docker, para servir las diferentes aplicaciones. También se desarrolló un gestor de alertas, capaz de enviar notificaciones a los dispositivos móviles de aquellos usuarios suscritos al bot de Telegram.

Para conocer en profundidad cómo interactúan los diferentes servicios entre sí, se recomienda ver el apartado 5.3.6.

5.1. Esquema general del mini servidor

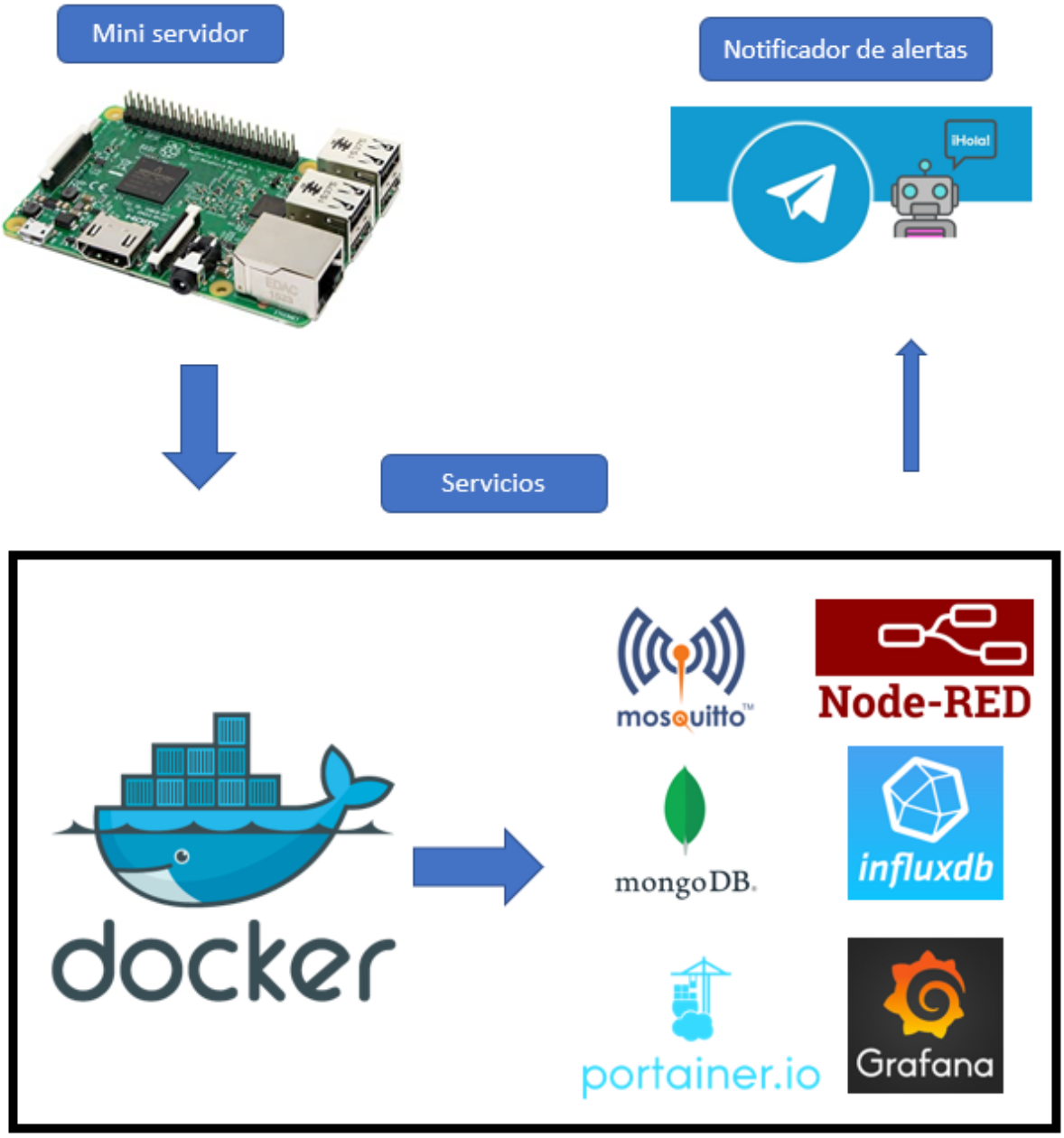


Figura 5.2 Esquema del mini servidor que monitoriza los datos de los diferentes nodos

Como se puede ver en la figura 5.2, el mini servidor implementado está formado por:

- **Una Raspberry Pi**, encargada de monitorizar los datos recibidos de los diferentes nodos (ESP32 y Raspberry Pi con cámara). El mini servidor arrancará sus servicios

en cuanto se le conecte con una fuente de alimentación y necesitará un minuto para estar plenamente operativo.

- **Docker**, software encargado de gestionar la monitorización, gracias a la cual se desplegaron un total de 6 servicios.
    - Para la gestión de los diferentes "contenedores" se usó Portainer.
    - Para la recepción de mensajes MQTT con transporte SSL/TLS se usó Eclipse-mosquitto.
    - Para procesar los datos se usó Node Red.
    - Para el almacenamiento de los datos se usó InfluxDB y MongoDB.
    - Para la visualización de los datos se usó Grafana.
- **Notificador de alertas**, el servicio Node Red, se encargará de la gestión de notificar las alertas de los niveles de CO2, temperatura y aforo cuando se superan sus respectivos límites. También, Node Red se encargó de la configuración de un bot en Telegram para poder llegar al mayor número de usuarios.

5.2. Docker

Docker es un software de código abierto que ayuda a automatizar el despliegue de aplicaciones/servicios. A través de "contenedores", los cuales son un método de virtualización que usa características de aislamiento de recursos del kernel de Linux, y de este modo evita tener que crear, mantener e iniciar múltiples máquinas virtuales. [57] .

Docker, partiendo de Linux, ha logrado simplificar y convertirse en una tecnología flexible y fácil de utilizar para trabajar con contenedores. [58]

La analogía con la industria del transporte es muy ilustrativa y muestra el camino futuro en el desarrollo de software, donde se diseñó un contenedor con medidas estándar, para poder ser transportado de una manera sencilla en barcos, trenes y camiones.

Aunque Docker se parezca a lo que hoy se conoce como la típica máquina virtual, posee algunas diferentes:

- Los contenedores Docker comparten recursos con el sistema operativo sobre el que se ejecutan, pudiendo arrancar o parar los contenedores rápidamente. A

diferencia de las máquinas virtuales como Vmware o VirtualBox, que se aíslan del sistema operativo sobre el que trabajan.

- La portabilidad de los contenedores, hace que los problemas de cambiar el entorno donde está corriendo la aplicación se reduzcan. De esta manera los contenedores Docker se centran en crear una aplicación que sea capaz de portar su contenido de forma sencilla.

Para poder usar Docker, habrá que instalarlo. En este enlace hay un tutorial de como instalar Docker [96].

5.2.1. Portainer

Portainer es una herramienta web de código abierto con la que podemos gestionar los contenedores de Docker, la infraestructura de soporte y todos los ajustes o implementaciones de Kubernetes, Docker Standalone y Docker Swarm, tanto de forma remota como local y con una interfaz más amigable que meter comandos por consola [59].

5.2.2. Docker Compose

Docker Compose es una herramienta de Docker que sirve para definir y ejecutar múltiples contenedores o servicios. Para ello se necesita un archivo de tipo YAML donde se especifica la configuración necesaria de los diferentes servicios necesarios para una aplicación, haciendo posible que ejecutando un simple comando "docker-compose up -d", se levanten todos los servicios configurados, sin necesidad de ir levantando un por uno [76].

Esto hace posible poder desplegar contenedores en cualquier dispositivo con Docker Compose instalado. Tutorial de instalación de Docker Compose [60] .

5.2.3. Docker Hub

Docker Hub [61] es un servicio proporcionado por Docker para buscar y compartir imágenes de contenedores. Docker Hub es el repositorio de imágenes de contenedores más grande del mundo, incluyendo desarrolladores oficiales y proyectos de código abierto.

Será necesario registrarse con un correo electrónico para poder acceder al contenido de Docker Hub, teniendo acceso a repositorios públicos gratuitos para almacenar y compartir imágenes.

Este servicio será utilizado en el proyecto para saber que configuración necesitaran los diferentes contenedores desplegados en el mini servidor.

5.3. Servicios del mini servidor en la Raspberry Pi

A continuación se pasará a explicar cada una de las herramientas y servicios utilizados en el mini servidor Raspberry Pi, que permiten procesar, almacenar y visualizar los datos recibidos por parte del ESP32 y la Raspberry Pi con cámara incorporada.

Para lograr que el mini servidor pueda cumplir con las tareas de procesar, almacenar y visualizar, se hará uso de la herramienta Docker [57] permitiendo desplegar las aplicaciones/servicios mostradas en la figura 5.3.

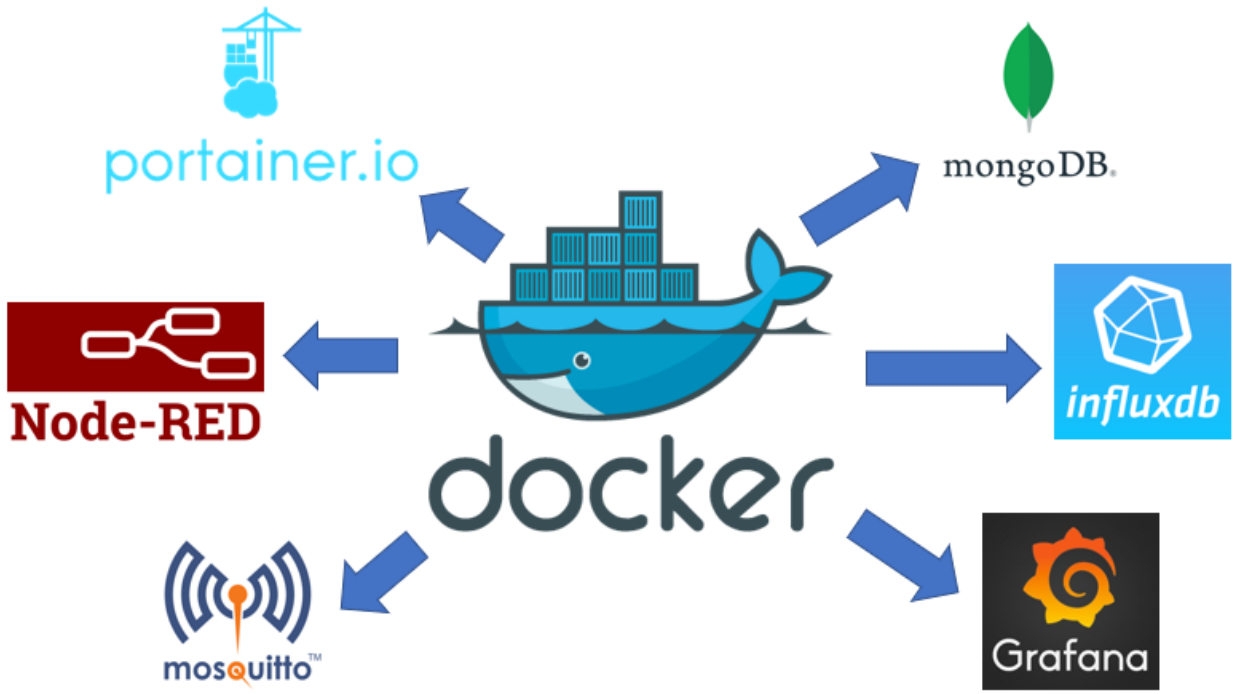


Figura 5.3 Visualización gráfica de como Docker sirve el resto de servicios del proyecto

La configuración de cada servicio en Docker Compose se explicará en sus respectivos apartados. El fichero que contiene todas las configuraciones se llama

"docker-compose.yml" y se encuentra en la carpeta "scripts adicionales". Este fichero será el encargado de descargar, configurar y poner en marcha los siguientes contenedores: Portainer, InfluxDB, Grafana, MongoDB, Eclipse-mosquitto y Node Red.

5.3.1. Portainer

Como se explicó en el apartado "5.2.1. Portainer". [59] Portainer servirá para la gestión de los diferentes contenedores utilizados en el mini servidor, pudiendo sortear la consola de comandos en el caso de querer actualizar, detener, iniciar o reiniciar algún contenedor.

Para acceder a la aplicación habrá que saber previamente la IP de las Raspberry Pi, pudiendo saberlo yendo a la consola de comandos e introduciendo el comando "ifconfig", se verá la información de la figura 5.4.

Figura 5.4 IP de la Raspberry Pi

Ahora se podrá ir al navegador web de cualquier dispositivo conectado a la misma red Wifi y teclear la IP del mini servidor, junto al puerto 9000, tal y como se muestra en la figura 5.5.

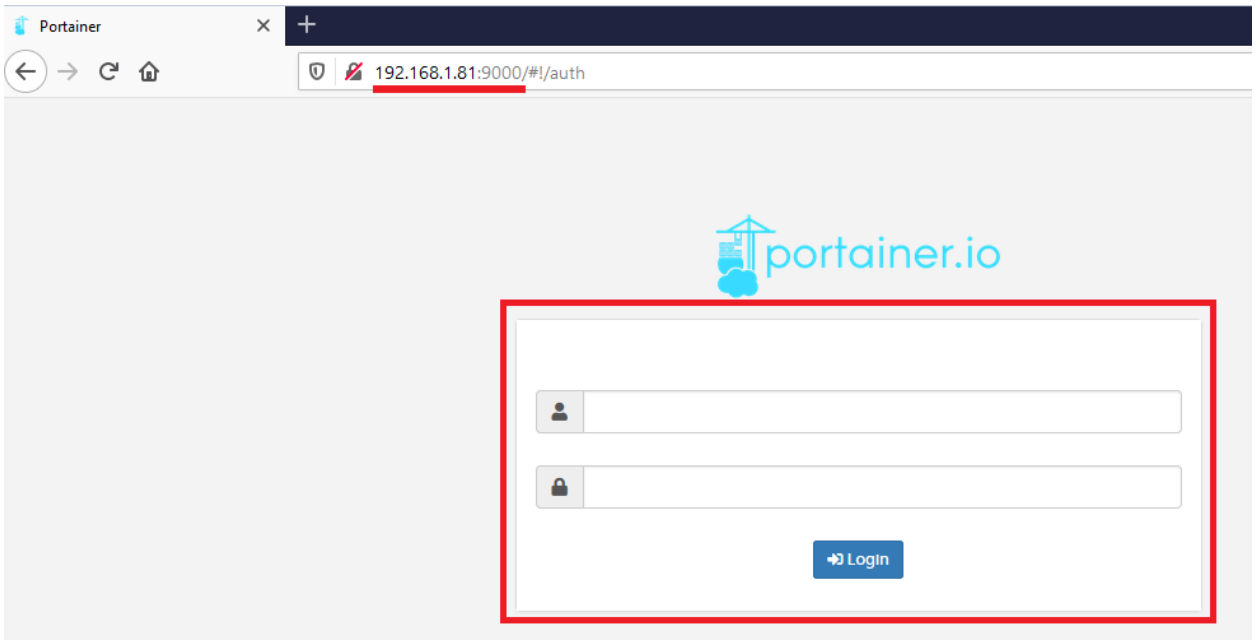


Figura 5.5 Portainer en su pantalla de Login

La primera vez que se conecte con Portainer este pedirá que se introduzca un usuario y contraseña nuevos [62] .

La configuración del contenedor de Portainer, para este proyecto, quedó especificada en fichero de Docker Compose [76] como se puede observar en la figura 5.6, donde se puede ver:

- **container_name**, para dar un nombre al contenedor de Portainer.
- **image**, para descargar la imagen de Portainer de Docker Hub [63] .
- **restart**, para reiniciar el contenedor siempre que se caiga o se arranque la Raspberry Pi.
- **volumes**, para indicar la ruta de las carpetas donde se compartirá información entre contenedor y Raspberry Pi, a la izquierda la carpeta que se encontrará en la Raspberry Pi y a la derecha la carpeta del contenedor.
- **ports**, para conectar los puertos del contenedor con los de la Raspberry Pi, se abren los indicados en la documentación de Docker Hub [64] . En este caso se habilitarán varios puertos, pero el que se usará será el puerto 9000, para conectarse desde cualquier navegador web, teniendo que identificarse con un usuario y contraseña.

```
########################## Portainer
# docker run --name portainer -it --restart=always -p 8000:8000 -p 9000:9000
# -v /var/run/docker.sock:/var/run/docker.sock
# -v /home/pi/Desktop/docker/portainer/data:/data portainer/portainer-ce
  portainer:
    container_name: portainer
    image: portainer/portainer-ce
    restart: always
    # Rutas donde se guardaran los datos
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/pi/Desktop/docker/portainer/data:/data
    ports:
      - 8000:8000
      - 9000:9000
```

Figura 5.6 Configuración del contenedor Portainer en Docker Compose

5.3.2. InfluxDB

InfluxDB [65] es una base de datos de series temporales de código abierto desarrollada por la empresa InfluxData [66] . Contando con una licencia MIT [67], siendo una licencia de software libre con restricciones mínimas sobre cómo se puede usar, modificar y redistribuir el software, incluyendo por lo general a una renuncia de garantía. [68]

InfluxDB está implementado en el lenguaje de programación Go [69], sirviendo para el almacenamiento y la recuperación de datos de series de tiempo en campos como monitoreo de operaciones, análisis en tiempo real y datos de sensores de IoT, haciendo que estas dos últimas características sean ideales para el proyecto desarrollado, contando con una potente sintaxis similar a SQL.

La configuración del contenedor de InfluxDB, para este proyecto, quedó especificada en fichero de Docker Compose [36] como se puede observar en la figura 5.7, donde se puede ver:

- **container_name**, para dar un nombre al contenedor de InfluxDB.
- **image**, para descargar la imagen de InfluxDB de Docker Hub, indicando la versión 1.8.

- **restart**, para reiniciar el contenedor siempre que se caiga o se arranque la Raspberry Pi.
- **environment**, para modificar variables de entorno de InfluxDB, donde se puede observar que [70] [71]:
    - Se habilita la autenticación del usuario para poder acceder a InfluxDB.
    - Se crea un super usuario con su contraseña.
    - Se crea una base de datos llamada "info_sensores".
    - Se crea un usuario, el cual tendrá permisos de lectura y escritura en la base de datos anteriormente creada.
    - Se indica la carpeta donde se guardarán los datos, tanto fuera como dentro del contenedor.
- **volumes**, para indicar la ruta de las carpetas donde se compartirá información entre contenedor y Raspberry Pi, a la izquierda la carpeta que se encontrará en la Raspberry Pi y a la derecha la carpeta del contenedor.
- **ports**, para conectar los puertos del contenedor con los de la Raspberry Pi, se abren los indicados en la documentación de Docker Hub [72] . En este caso se habilitarán varios puertos, de los cuales se usará para la comunicación entre InfluxDB, Node Red y Grafana el puerto 8086.

```
######################### InfluxDB
# docker run --name influxdb -it --restart=always -p 8086:8086 -p 8083:8083 -p 2003:2003
# -v /home/pi/Desktop/docker/influxdb/var/lib/influxdb:/var/lib/influxdb
# -v /home/pi/Desktop/docker/influxdb/etc/influxdb:/etc/influxdb influxdb:1.8
influxdb:
  container_name: influxdb
  image: influxdb:1.8
  restart: always
  # Le ponemos contraseña a la BBDD
  environment:
    INFLUXDB_HTTP_AUTH_ENABLED: 'true'
    INFLUXDB_ADMIN_USER: admin
    INFLUXDB_ADMIN_PASSWORD: secret
    INFLUXDB_DB: info_sensores
    INFLUXDB_USER: user
    INFLUXDB_USER_PASSWORD: secret2
    # Dando permisos de lectura y escritura
    INFLUXDB_READ_USER: user
    INFLUXDB_WRITE_USER: user
    INFLUXDB_DATA_DIR: /var/lib/influxdb/data
  # Rutas donde se guardaran los datos
  volumes:
    - /home/pi/Desktop/docker/influxdb/var/lib/influxdb:/var/lib/influxdb
    #- /home/pi/Desktop/docker/influxdb/etc/influxdb/:/etc/influxdb/
  ports:
    - 8086:8086
    - 8083:8083
    - 2003:2003
```

Figura 5.7 Configuración del contenedor InfluxDB en Docker Compose

5.3.3. Grafana

Grafana [73] [74] es un software libre desarrollado en lenguaje Go con licencia de Apache 2.0 [75], siendo capaz de permitir la visualización de datos métricos, mediante la creación de cuadros de mando a partir de múltiples fuentes, incluyendo bases de datos de series temporales como InfluxDB.

La configuración del contenedor de Grafana, para este proyecto, quedó especificada en fichero de Docker Compose [76] como se puede observar en la figura 5.8, donde se puede ver:

- **container_name**, para dar un nombre al contenedor de Grafana.
- **image**, para descargar la imagen de Grafana de Docker Hub [63] .

- **restart**, para reiniciar el contenedor siempre que se caiga o se arranque la Raspberry Pi.
- **volumes**, para indicar la ruta de las carpetas donde se compartirá información entre contenedor y Raspberry Pi, a la izquierda la carpeta que se encontrará en la Raspberry Pi y a la derecha la carpeta del contenedor.
- **ports**, para conectar los puertos del contenedor con los de la Raspberry Pi, se abren los indicados en la documentación de Docker Hub [77] . En este caso se usará el puerto 3000, usado por defecto en Grafana.

```
######################### Grafana
# docker run --name grafana -it --user $(id -u)
# --restart=always -p 3000:3000
# -v /home/pi/Desktop/docker/grafana/var/lib/grafana:/var/lib/grafana grafana/grafana
grafana:
  container_name: grafana
  image: grafana/grafana
  restart: always
  user: "1000"
  # Rutas donde se guardaran los datos
  volumes:
    - /home/pi/Desktop/docker/grafana/var/lib/grafana:/var/lib/grafana
  ports:
    - 3000:3000
```

Figura 5.8 Configuración del contenedor Grafana en Docker Compose

Para acceder al contenedor Grafana poner la IP de la Raspberry Pi junto al puerto 3000, como se puede ver en la figura 5.9, donde se pedirá un usuario y contraseña, que por defecto serán:

- Usuario: admin
- Contraseña: admin

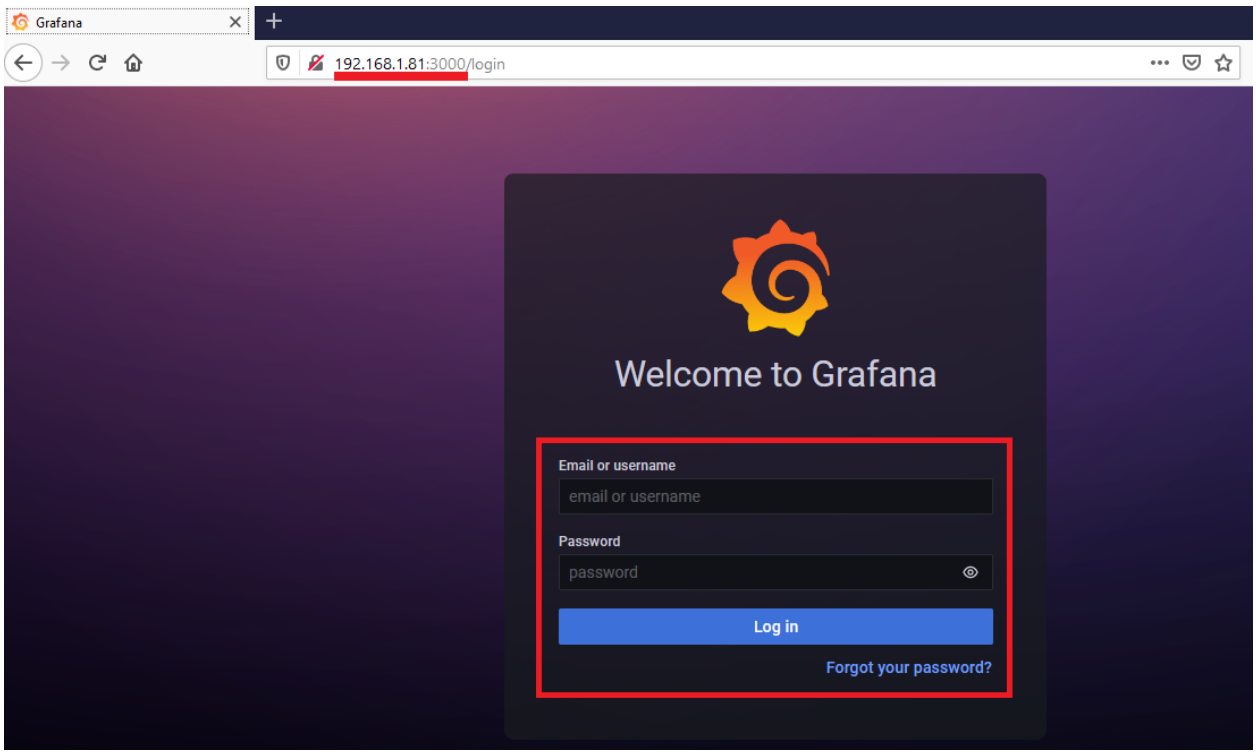


Figura 5.9 Grafana en su pantalla de Login

Seguidamente se pedirá que se cambie la contraseña, para una mayor seguridad.

5.3.4. MongoDB

MongoDB [78] es un sistema de base de datos NoSQL, orientado a documentos y de código abierto. En lugar de guardar los datos en tablas como hacen las bases de datos relacionales, MongoDB guarda estructuras de datos BSON [79], una especificación similar a JSON, con un esquema dinámico haciendo que la integración de datos sea más fácil y rápida.

MongoDB está implementado usando C++ y cuenta con una licencia GNU AGPL v3.0 [80]. En concreto en este proyecto se utilizará la versión "bionic" dado que es una versión que funciona en arquitecturas ARM como la utilizada en Raspberry Pi. [81]

La configuración del contenedor de MongoDB, para este proyecto, quedó especificada en el fichero de Docker Compose [76] como se puede observar en la figura 5.10, donde se puede ver:

- **container_name**, para dar un nombre al contenedor de MongoDB.

- **image**, para descargar la imagen de MongoDB de Docker Hub [63], concretamente la versión "bionic".
- **restart**, para reiniciar el contenedor siempre que se caiga o se arranque la Raspberry Pi.
- **environment**, para modificar variables de entorno de MongoDB, donde se puede observar que:
    - Se crea un super usuario con su contraseña.
    - Se crea una base de datos llamada "telegram".
    - Se crea un usuario, el cual tendrá permisos de lectura y escritura en la base de datos anteriormente creada.
- **volumes**, para indicar la ruta de las carpetas donde se compartirá información entre contenedor y Raspberry Pi, a la izquierda la carpeta que se encontrará en la Raspberry Pi y a la derecha la carpeta del contenedor.

  Se ve que hay un código bash comentado en esta sección, el cual tendrá que ser utilizado para crear un fichero .bash llamado "mongo-init.sh" en la ruta señala en el "volumes", ya que se usará para asignar el nuevo usuario creado, en la sección de "environment", a la base de datos "telegram" y seguidamente se creará la colección "suscripciones", donde se guardaran los "id_chats" de los usuarios que decidan suscribirse a las alarmas del sistema. [82] [83]

- **ports**, para conectar los puertos del contenedor con los de la Raspberry Pi, se abren los indicados en la documentación de Docker Hub [84]. En este caso se usará el puerto 27017.

```
######################### MongoDB
  mongodb:
    container_name: mongodb
    image: mongo:bionic
    # Primero ejecutar sin este comando, generando los los usuarios
    restart: always
    # Le ponemos contraseña a la BBDD
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: secret
      MONGO_INITDB_DATABASE: telegram
      MONGO_INITDB_USERNAME: user
      MONGO_INITDB_PASSWORD: secret2
    # Rutas donde se guardaran los datos
    volumes:
      # Importante este fichero crearlo con este formato
      # #!/bin/bash
      # mongo -- "$MONGO_INITDB_DATABASE" <<EOF
      #     var database = db.getSiblingDB('$MONGO_INITDB_DATABASE');
      #     database.createUser({ user: '$MONGO_INITDB_USERNAME',
      #                           pwd: '$MONGO_INITDB_PASSWORD',
      #                           roles: [ { role: "readWrite", db: '$MONGO_INITDB_DATABASE' } ]});
      #     database.createCollection('suscripciones');
      # EOF
      - /home/pi/Desktop/docker/mongodb/mongo-init.sh:/docker-entrypoint-initdb.d/mongo-init.sh:ro
      - /home/pi/Desktop/docker/mongodb/data:/data/db
    ports:
      - 27017:27017
```

Figura 5.10 Configuración del contenedor MongoDB en Docker Compose

MongoDB será utilizado para guardar y eliminar los "id_chats" de los usuarios del bot en Telegram que decidan suscribirse a las notificaciones de las alarmas.

5.3.5. Eclipse-mosquitto

Eclipse-mosquitto [85] es un bróker de mensajería de código abierto con licencia EPL/EDL [86], que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT. Mosquitto es liviano y adecuado para usar en todo tipo de dispositivos.

Para este proyecto, Mosquitto será utilizado para la realización de comunicaciones vía MQTT con transporte SSL/TLS entre ESP32(pública), Raspberry Pi con cámara (pública) y la Raspberry Pi que hará de mini servidor(suscribe).

La configuración del contenedor de Eclipse-mosquitto, para este proyecto, quedó especificada en el fichero de Docker Compose [76] como se puede observar en la figura 5.11, donde se puede ver:

- **container_name**, para dar un nombre al contenedor de Eclipse-mosquitto.
- **image**, para descargar la imagen de Eclipse-mosquitto de Docker Hub [63], utilizando la versión 1.6.10.
- **restart**, para reiniciar el contenedor siempre que se caiga o se arranque la Raspberry Pi.
- **volumes**, para indicar la ruta de las carpetas donde se compartirá información entre contenedor y Raspberry Pi, a la izquierda la carpeta que se encontrará en la Raspberry Pi y a la derecha la carpeta del contenedor.

  Concretamente para habilitar la comunicación vía MQTT con transporte SSL/TLS, se añadirá una carpeta con la configuración vista en esta página [87], que explica cómo configurar el contenedor de Eclipse-mosquitto.

- **ports**, para conectar los puertos del contenedor con los de la Raspberry Pi, se abren los indicados en la documentación de Docker Hub [88] , donde se habilitará el puerto 8883 que será el que permite el uso de TLS con MQTT.

```
########################## Broker MQTT mosquitto
mosquitto:
  container_name: mosquitto_mqtt
  image: eclipse-mosquitto:1.6.10
  restart: always
  # Rutas donde se guardaran los datos
  volumes:
    - /home/pi/Desktop/docker/mosquitto/config:/mosquitto/config/
  ports:
    - 1883:1883
    - 8883:8883
```

Figura 5.11 Configuración del contenedor Eclipse-mosquitto en Docker Compose

5.3.6. Node Red

Node Red [89] es una herramienta de programación para conectar dispositivos de hardware, API y servicios en línea de forma fácil y sencilla.

Proporciona un editor basado en navegador web, que se puede utilizar para crear funciones de Javascript.

Los flujos creados en Node Red se almacenan en formato JSON y posee una licencia Apache 2.0 [90].

Para poder instalar y usar los diferentes nodos para el proyecto se siguió este tutorial [91].

Los paquetes de módulos usados en el proyecto son los que se pueden observar en la figura 5.12.

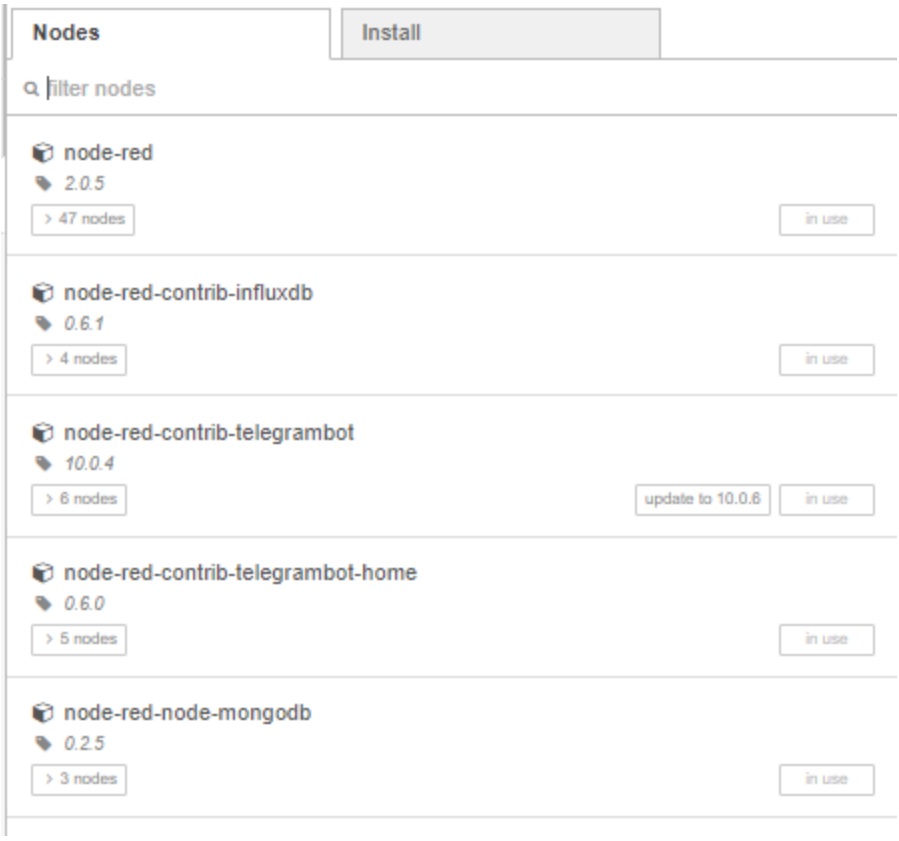


Figura 5.12 Paquetes de Node Red usados en el proyecto

- **node-red-contrib-influxdb**, para trabajar con InfluxDB.
- **node-red-contrib-telegrambot** y **node-red-contrib-telegrambot-home**, para poder configurar el bot de Telegram, que permitirá enviar alertas.
- **node-red-contrib-mongodb**, para trabajar con MongoDB.

Ya que Node Red puede ser accedido desde un navegador web introduciendo la IP de la Raspberry Pi junto al puerto 1880, como se ilustra en la figura 5.13 y, por defecto, viene sin una pantalla de login. Investigando se encontró la manera de hacer que el

usuario tuviera que identificarse introduciendo un usuario y contraseña, se recomienda consultar la referencia [92].

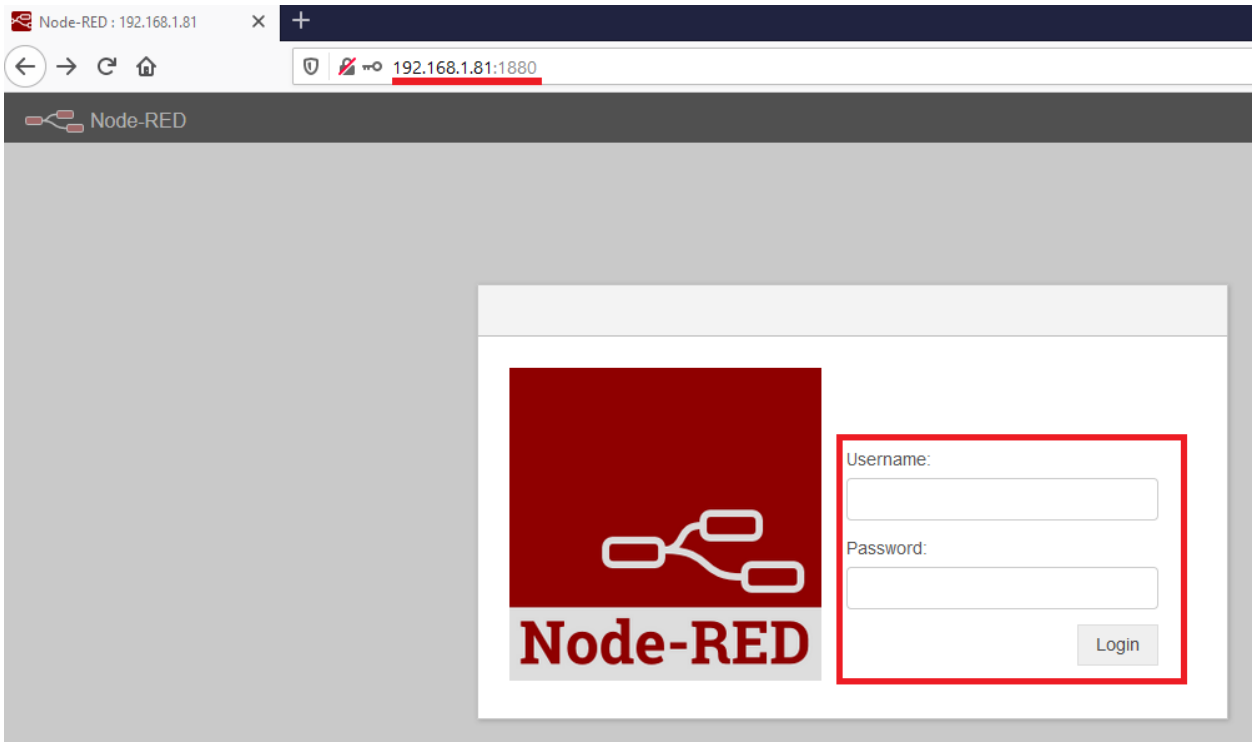


Figura 5.13 Node Red con pantalla de login

La configuración del contenedor de Node Red, para este proyecto, quedó especificada en el fichero de Docker Compose [76] como se puede observar en la figura 5.14, donde se puede ver:

- **container_name**, para dar un nombre al contenedor de Node Red.
- **image**, para descargar la imagen de Node Red de Docker Hub [63] .
- **restart**, para reiniciar el contenedor siempre que se caiga o se arranque la Raspberry Pi.
- **volumes**, para indicar la ruta de las carpetas donde se compartirá información entre contenedor y Raspberry Pi, a la izquierda la carpeta que se encontrará en la Raspberry Pi y a la derecha la carpeta del contenedor.
- **ports**, para conectar los puertos del contenedor con los de la Raspberry Pi, se abren los indicados en la documentación de Docker Hub [93], donde se habilitará el puerto 1880, que es el usado por defecto en Node Red.

```
######################### Node Red
# docker run --name nodered -itd --restart=always -p 1880:1880
# -v /home/pi/Desktop/docker/nodered/data:/data nodered/node-red
nodered:
  container_name: nodered
  image: nodered/node-red
  restart: always
  # Rutas donde se guardaran los datos
  volumes:
    - /home/pi/Desktop/docker/nodered/data:/data
  ports:
    - 1880:1880
```

Figura 5.14 Configuración del contenedor Node Red en Docker Compose

A continuación se pasará a hablar de la figura 5.15 donde se puede ver las interacciones entre las diferentes herramientas de forma gráfica.

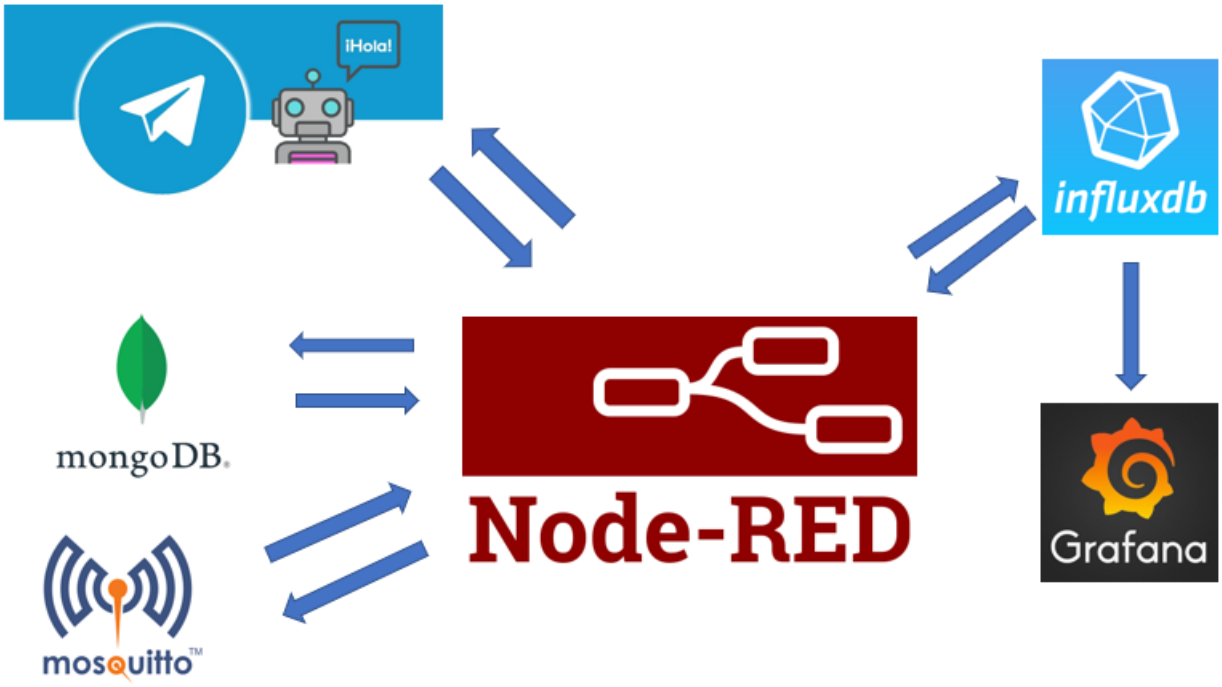


Figura 5.15 Interacción de las diferentes aplicaciones entre sí

Como se puede observar Node Red será la piedra angular en la gestión de las interacciones entre las distintas aplicaciones que componen el proyecto. Para ser explicado de manera clara se dividirán entre 3 flujos distintos:

5.3.6.1. Flujo de "Suscripción MQTT"

Este flujo de Node Red se configuró para procesar y almacenar en InfluxDB los datos enviados vía MQTT con transporte SSL/TLS por el ESP32 y la Raspberry Pi con cámara.

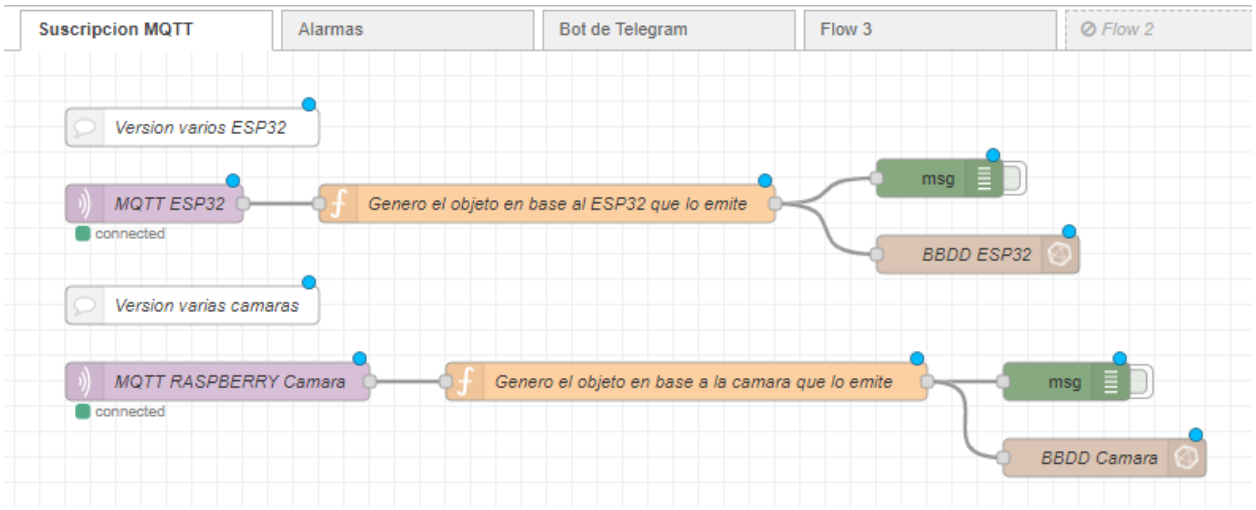


Figura 5.16 Vista general del flujo "Suscripción MQTT"

Como se puede ver en la figura 5.16, se utilizaron los siguientes nodos:

- **mqtt in**, es un nodo que permite configurar un bróker MQTT para que sea capaz de recibir un topic, el cual se especifica en el propio nodo. También, se puede indicar la calidad de la conexión (QoS). La configuración del bróker se puede ver en las figuras 5.17 y 5.18.

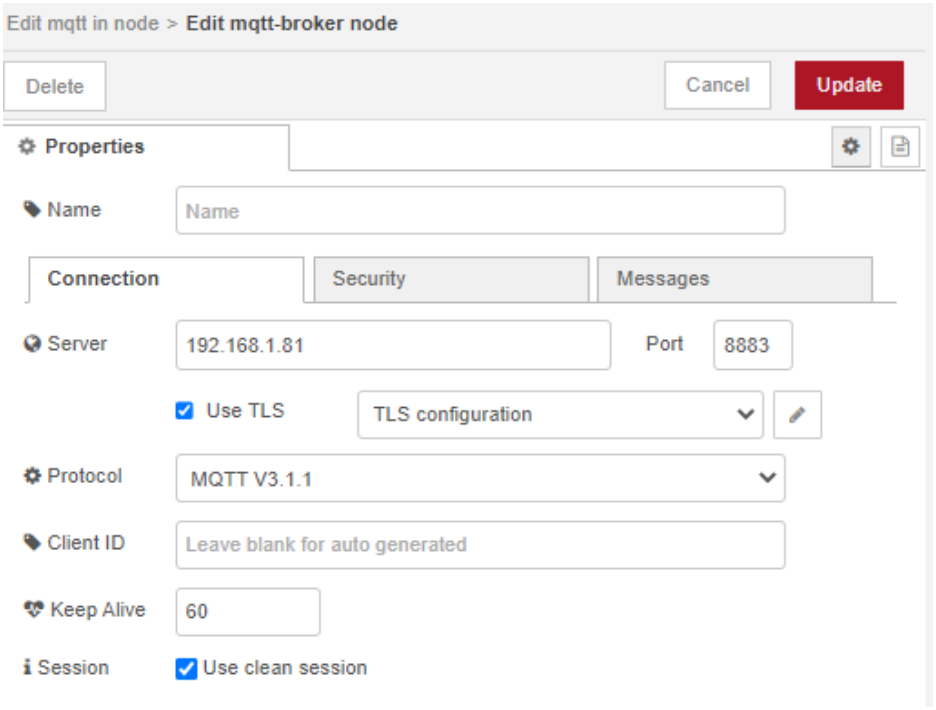


Figura 5.17 Configuración de un bróker MQTT en el nodo "mqtt in"

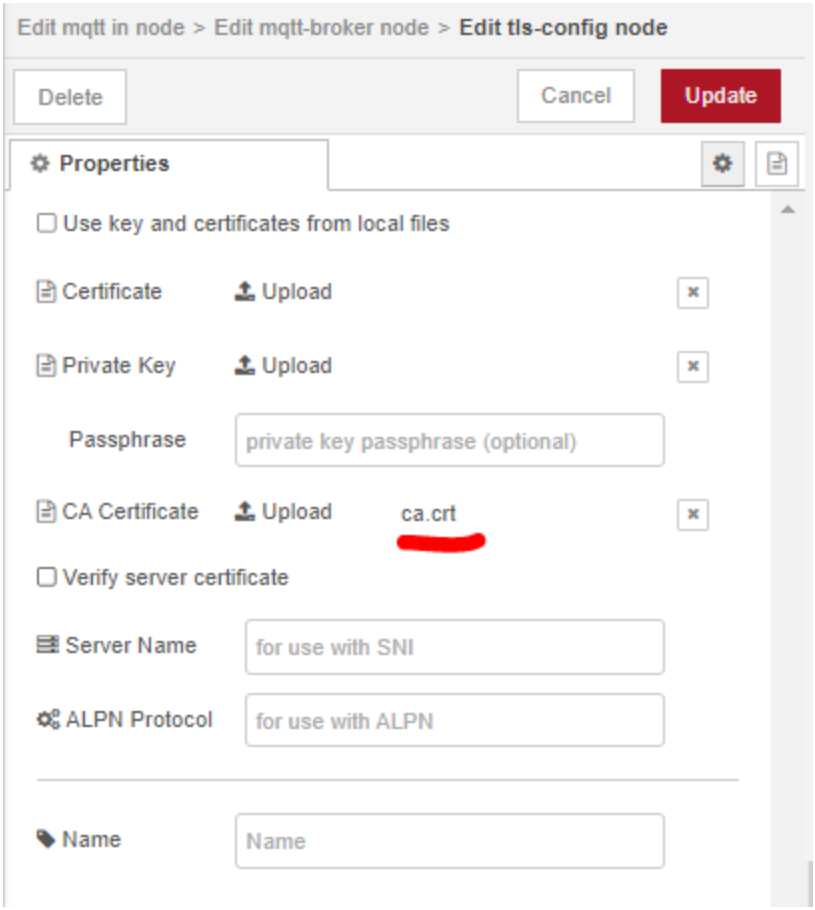


Figura 5.18 Configuración de seguridad TLS en el nodo "mqtt in", donde se añade el certificado "ca.crt", generado con este script

- **function**, es un nodo que permite desarrollar pequeños programas para procesar y preparar los datos para su almacenamiento en InfluxDB.
- **debug**, es un nodo que muestra por la pestaña "Debug message" de Node Red, la estructura y contenido de los datos que se están tratando en el flujo.
- **influxdb out**, es un nodo que permite configurar un servidor que tenga una instalación de InfluxDB, como se puede observar en la figura 5.19, donde la IP es la de la Raspberry Pi que tenga, en este caso, instalado el contenedor InfluxDB.

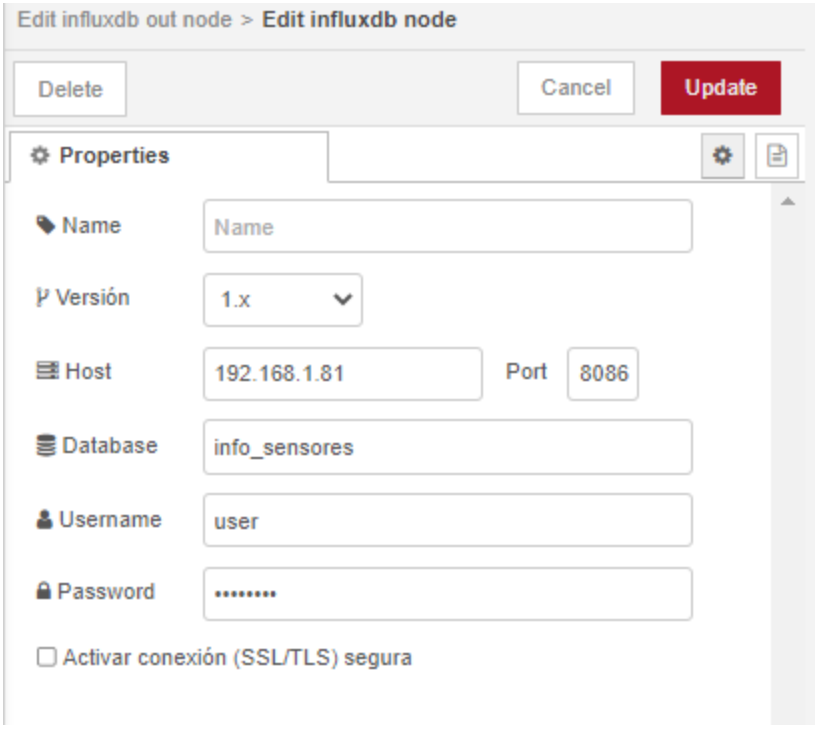


Figura 5.19 Configuración de un servidor InfluxDB para el nodo "influxdb out"

5.3.6.2. Flujo de "Alarmas"

Este flujo de Node Red se configuró para consultar las BBDD InfluxDB y MongoDB cada 5 minutos, donde si se detecta que se sobrepasan los límites configurados de CO2, temperatura y aforo, se enviarán alertas a los usuarios suscritos al bot de Telegram "@esp32_ucm_bot".

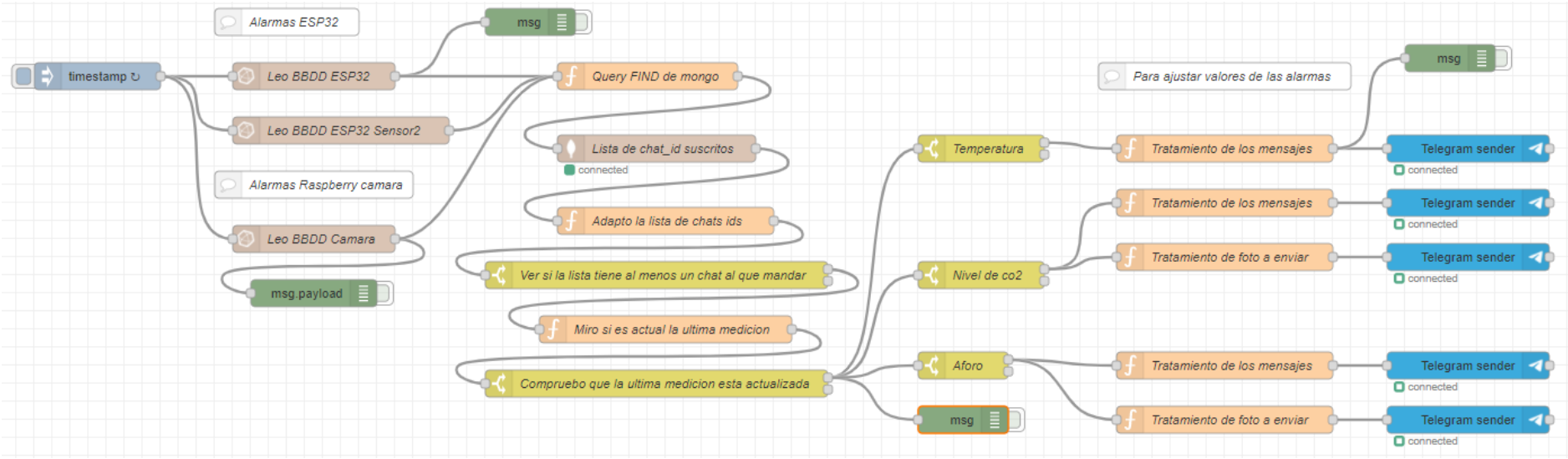


Figura 5.20 Vista general del flujo "Alarmas"

Como se puede ver en la figura 5.20, se utilizaron los siguientes nodos:

- **timestamp**, es un nodo que permite repetir ejecuciones del flujo indefinidamente, en el proyecto se usará, para que cada 5 minutos compruebo los niveles de CO2, temperatura y aforo.
- **debug**, es un nodo que muestra por la pestaña "Debug message" de Node Red, la estructura y contenido de los datos que se están tratando en el flujo.
- **switch**, es un nodo que cuando llega un mensaje, evaluará cada una de las condiciones definidas dentro de él y redireccionará el mensaje a la salida correspondiente. Se usará para comprobar que no se sobrepasan los límites de los niveles de CO2, temperatura y aforo.
- **function**, es un nodo que permite desarrollar pequeños programas para:
    - Hacer consultas en las bases de datos de InfluxDB y MongoDB
    - Personalizar las alertas enviadas al bot de Telegram
- **influxdb in**, es un nodo que permite realizar consultas básicas a la base de datos de series temporales InfluxDB. En el proyecto se utilizará para consultar los últimos valores recibidos de los sensores, tanto del ESP32 como de la Raspberry Pi con cámara.
- **mongodb in**, es un nodo que permite realizar consultas básicas a la base de datos de MongoDB. En el proyecto se utilizará para consultar los "id_chats" de los usuarios suscritos al bot de Telegram para recibir las alertas.
- **sender**, es un nodo que permite comunicarse con un bot de Telegram, siguiendo esta estructura determinada en el campo "msg.payload":
    - **content**, contenido del mensaje.
    - **type**, el tipo del contenido del mensaje. Por ejemplo: texto, fotos, videos, etc.
    - **chatId**, donde se podrá poner un array de chat ids.

  En el proyecto se usará para enviar textos e imágenes cuando salten las alarmas de los niveles de CO2, temperatura y alerta. Ver la configuración de este nodo en la figura 5.21.

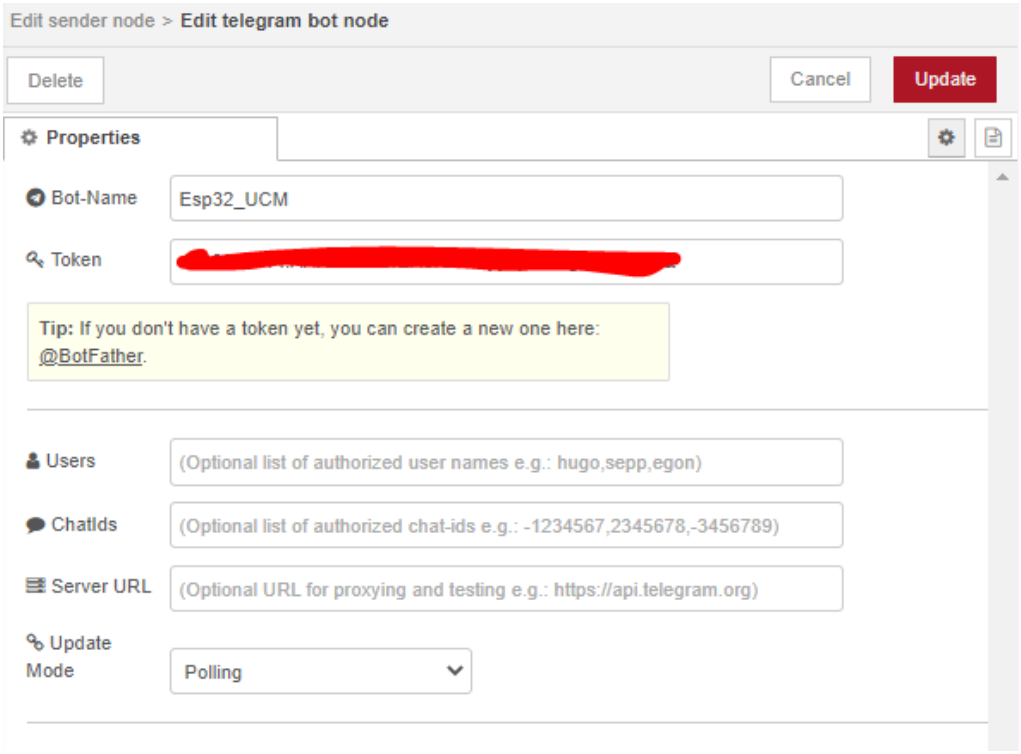


Figura 5.21 Configuración del bot de Telegram, censurando el "Token"

5.3.6.3. Flujo de "Bot de Telegram"

Este flujo de Node Red se configuró para habilitar y gestionar los comandos del bot de Telegram "@esp32_ucm_bot", así como la suscripción y cancelación a las alertas del sistema.

Los comandos son los siguientes:

- **Start**

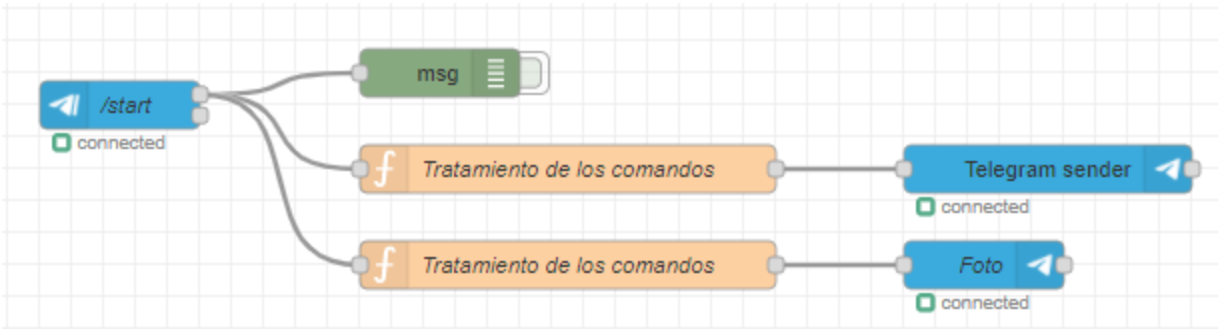


Figura 5.22 Aspecto del comando "/start"

- **Help**

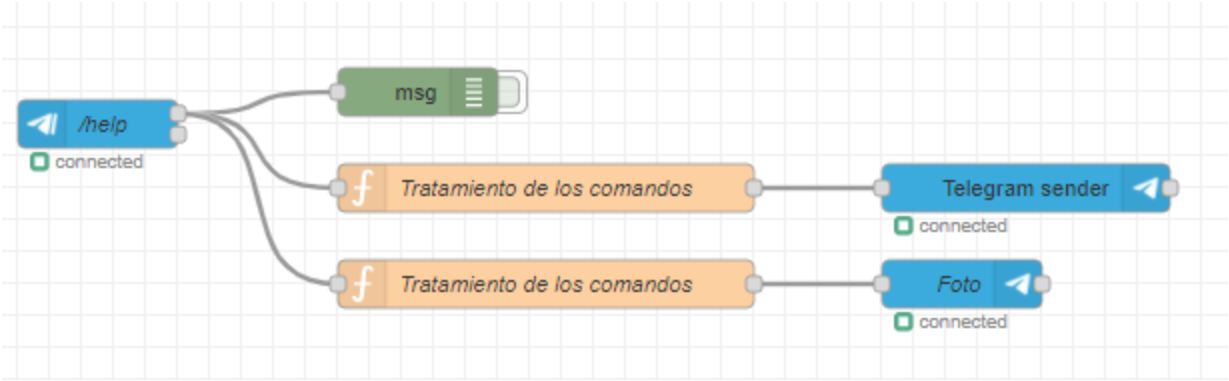


Figura 5.23 Aspecto del comando "/help"

- **Sub y cancel**

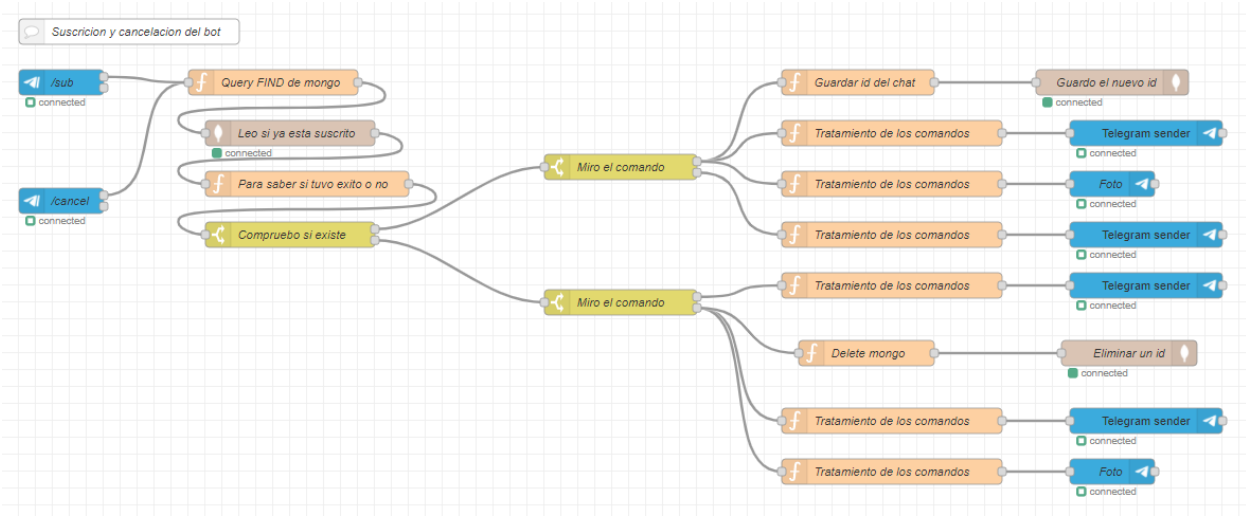


Figura 5.24 Aspecto de los comandos "/sub" y "/cancel"

Como se puede ver en las figuras 5.22, 5.23 y 5.24, se utilizaron los siguientes nodos:

- **debug**, es un nodo que muestra por la pestaña "Debug message" de Node Red, la estructura y contenido de los datos que se están tratando en el flujo.
- **switch**, es un nodo que cuando llega un mensaje, evaluará cada una de las condiciones definidas dentro de él y redireccionará el mensaje a la salida correspondiente. Se usará para comprobar:
    - Si el usuario ya está suscrito al sistema de alertas.
    - Si el comando utilizado es: "/sub" o "/cancel".

- **function**, es un nodo que permite desarrollar pequeños programas para:
    - Hacer consultas en la base de datos de MongoDB
    - Personalizar los mensajes y fotos enviadas al bot de Telegram
- **mongodb in**, es un nodo que permite realizar consultas básicas a la base de datos de MongoDB. En el proyecto se utilizará para consultar si los "id_chats" de los usuarios que solicitan suscribirse o desuscribirse(cancel) al sistema de envío de alertas se encuentran registrados en la base de datos de MongoDB.
- **mongodb out**, es un nodo que permite realizar consultas básicas a la base de datos de MongoDB, para crear o eliminar datos. En el proyecto se utilizara para registrar o eliminar el "id_chat" del usuario que pida suscribirse o desuscribirse del sistema de alertas.
- **command**, es un nodo de Telegram que se activa cuando se recibe un comando en el chat del bot en Telegram. En el proyecto se usará para poder saber cuando se escriben los comandos: "/start", "/help", "/sub" y "/cancel".
- **sender**, es un nodo que permite comunicarse con un bot de Telegram, siguiendo esta estructura determinada en el campo "msg.payload":
    - **content**, contenido del mensaje.
    - **type**, el tipo del contenido del mensaje. Por ejemplo: texto, fotos, videos, etc.
    - **chatId**, donde se podrá poner un array de chat ids.

  En el proyecto se usará para enviar textos e imágenes de confirmación cuando el usuario interactúe con los comandos implementados.

Para poder saber cómo utilizar los nodos de Telegram en Node Red se hizo uso de un blog en Internet [94].

5.3.6.4. Creación de un bot en Telegram

Telegram [95] es una plataforma de mensajería instantánea, envío de varios archivos y la comunicación en masa. Conocida por su gran privacidad y seguridad a la hora del envío de mensajes.

Telegram ofrece entre sus funcionalidades, creación de grupos con gran capacidad, creación de chat "secretos" donde se puede elegir que se autodestruyen los mensajes enviados pudiendo modificar el tiempo que se deja al receptor para visualizar los

mensajes, fotos, videos, etc. También cuenta con la capacidad de realizar videoconferencias y llamadas, así como un apartado para "mensajes guardados", donde se puede almacenar conversaciones, apuntes personales y recordatorios.

Para la automatización de tareas masivas Telegram ofrece la creación de bots, que pueden realizar actividades y servicios extra como pagos, juegos, moderación de grupos o un sistema de alarmas para la medición de la calidad del aire.

Telegram posee en la actualidad más de 500 millones de usuario activos mensuales, lo que la hace una herramienta a tener en cuenta.

En este proyecto fue utilizada para la creación de un bot para que pueda transmitir las alertas a los usuarios suscritos.

A continuación se mostrará un breve tutorial de como crear un bot en Telegram.

1. Habrá que ir a la aplicación de Telegram, si no se tiene cuenta, habrá que crearla.
2. Después buscar en la barra de búsqueda de Telegram "Bot Father", cómo se enseña en la figura 5.25.

Figura 5.25 Buscando BotFather

3. Se le da click al chat y al pulsar "start" o introduciendo "/help" se verá un mensaje que indica cómo crear un nuevo bot, como se muestra en la figura 5.26.

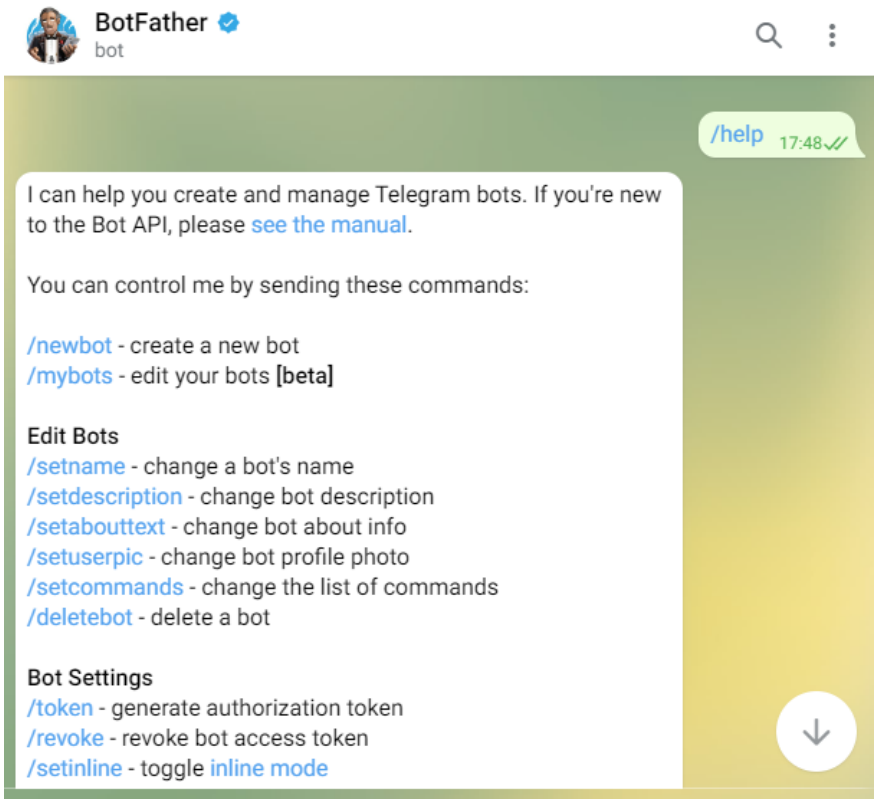


Figura 5.26 Introduciendo el comando "/help"

4. Teclear el comando "/newbot" y pedirá un nombre para el bot y, luego, pedirá que se elija un nombre de usuario para el bot. Finalizando el proceso y mostrando el "token" del bot, mediante el cual se podrá trabajar con el bot en Node Red. Todo este proceso se puede ver en la figura 5.27.

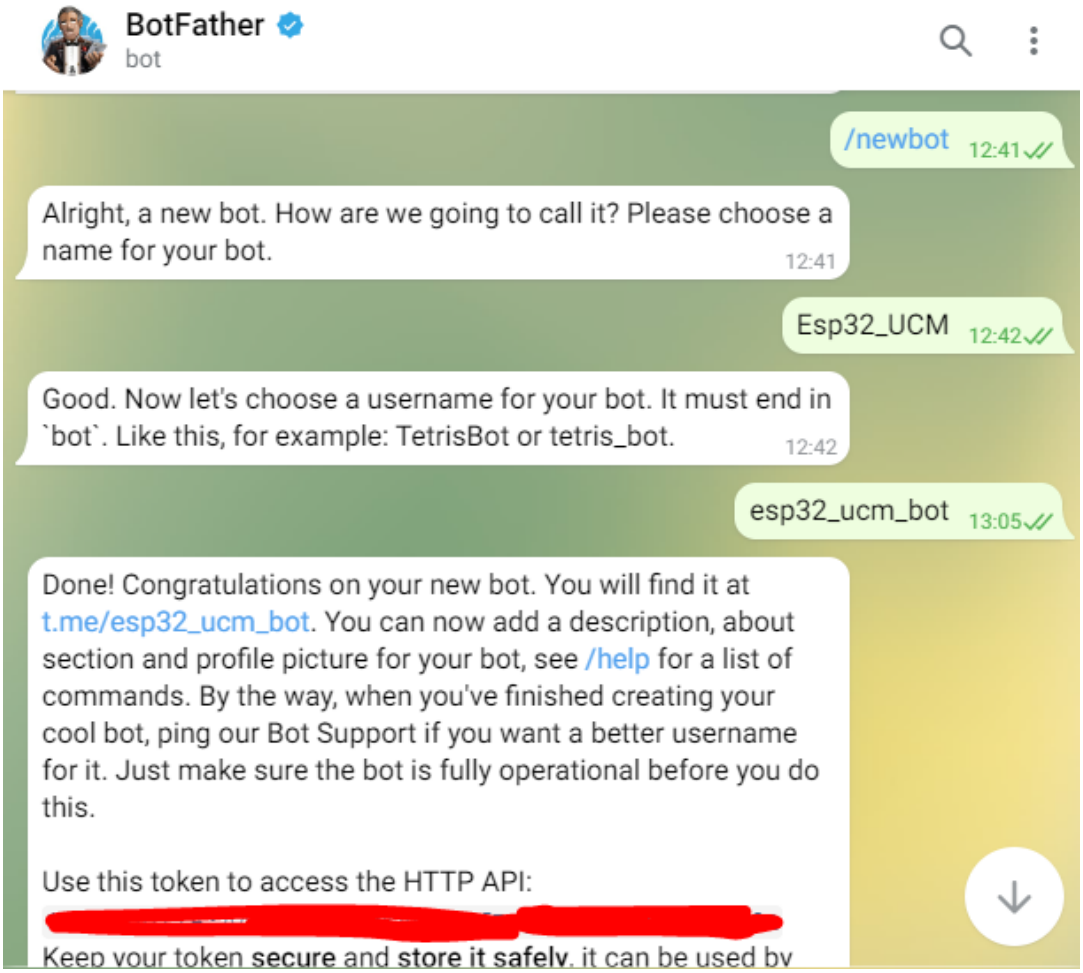


Figura 5.27 Proceso de creación de un bot

5. Si se quiere añadir una foto de perfil al bot, se tendría que introducir el comando "/setuserpic"(los comandos pueden cambiar con el tiempo, siempre consultarlos con "/help"), elegir el bot al que se le va a cambiar la foto de perfil y enviar la foto que se desee, como se indica en la figura 5.28.

Figura 5.29 Proceso para modificar la foto de perfil del bot

6. A partir de aquí ya se podrá acceder al bot creado, yendo a la barra del buscador de Telegram y poniendo el nombre del bot o el nombre de usuario, como se observa en la figura 5.30.

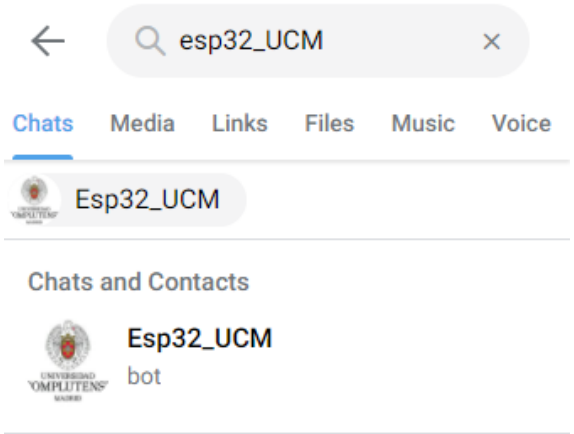


Figura 5.30 Búsqueda del bot creado

5.4. Modelos de datos

En este apartado se hablará de la estructura interna de las bases de datos InfluxDB y MongoDB, habiendo escogido estos dos tipos de BBDD para poder almacenar los datos del proyecto de una forma ordenada y fácil de consultar.

5.4.1. Base de datos InfluxDB

Como se puede observar en la figura 5.31, el modelo entidad relación consta de dos entidades independientes, llamadas en InfluxDB "mediciones" o "measurements".

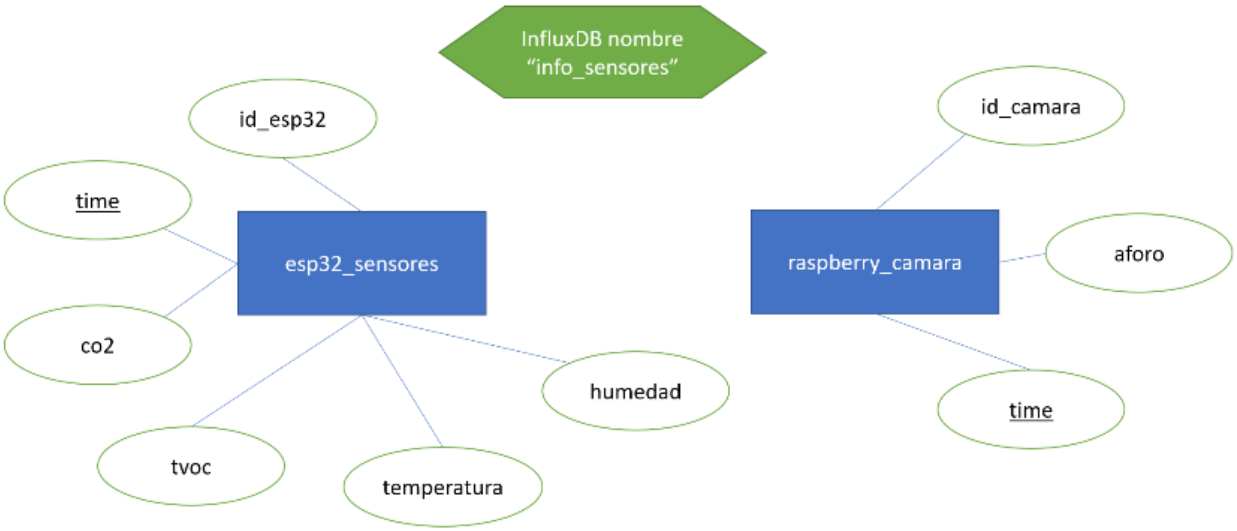


Figura 5.31 Modelo entidad-relación de InfluxDB

A continuación, se pasará a describir qué significa cada uno de los atributos de las "mediciones" o "measurements":

Medición "esp32_sensores"

En esta "medición", se almacenarán los datos relativos a las mediciones de los sensores SGP30 y Si7021.

- **id_esp32**, servirá para identificar a cada uno de los microcontroladores ESP32 que envíen datos. Este campo será de tipo string.

- **time**, será la clave de la "medición" y será un campo donde se registre la fecha y hora de la inserción de los datos en la "medición". Este campo será de tipo timestamp.
- **co2**, será el campo usado para guardar el nivel de CO2 medido por el sensor SGP30. Este campo será de tipo integer.
- **tvoc**, será el campo usado para guardar el nivel de TVOC medido por el sensor SGP30. Este campo será de tipo integer.
- **temperatura**, será el campo usado para guardar el nivel de temperatura medido por el sensor Si7021. Este campo será de tipo float.
- **humedad**, será el campo usado para guardar el nivel de humedad medido por el sensor Si7021. Este campo será de tipo float.

Medición "raspberry_camara"

En esta "medición", se almacenarán los datos relativos a las mediciones de la Raspberry Pi con cámara encargada de la medición del aforo en un espacio cerrado.

- **id_camara**, servirá para identificar a cada una de las Raspberry Pi con una cámara que envíen datos. Este campo será de tipo integer.
- **time**, será la clave de la "medición" y será un campo donde se registre la fecha y hora de la inserción de los datos en la "medición". Este campo será de tipo timestamp.
- **aforo**, será el campo usado para guardar el nivel de aforo medido por la Raspberry Pi con cámara. Este campo será de tipo integer.

5.4.2. Base de datos MongoDB

Como se puede observar en la figura 5.32, el modelo entidad relación consta de una entidad, llamada en MongoDB "colección" o "collection".

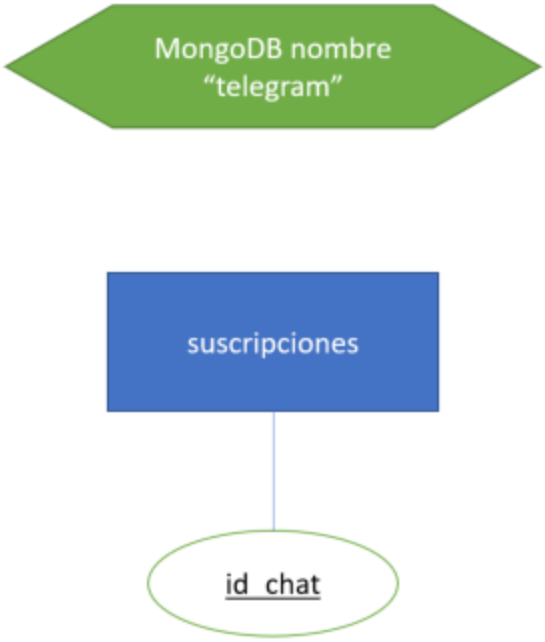


Figura 5.32 Modelo entidad-relación de MongoDB

A continuación, se pasará a describir qué significa cada uno de los atributos de la “colección” o “collection”:

Colección “suscripciones”

En esta “colección”, se almacenarán los datos relativos a los identificadores de chat que Telegram asigna a las personas que se conectan al bot de Telegram.

- **id_chats**, será la clave de la colección y servirá para identificar a cada uno de los usuarios que se suscriban al sistema de alertas del bot de Telegram. Este campo será de tipo integer.

5.5. Visualización de los datos y notificación de alarmas

5.5.1. Panel de control de Grafana

Grafana proporciona diferentes formas de visualizar los datos. A continuación se mostrará un panel de control en la figura 5.33 que es una mera demostración de cómo visualizar los datos más relevantes del proyecto, no siendo la única forma de visualizarlos, ya que dependerá de las necesidades del usuario.

Figura 5.33 Aspecto general del panel de control en Grafana

En la figura 5.34 se resaltan algunos campos de interés a la hora de visualizar los datos:

- **La línea azul**, arriba a la derecha, indica los campos que pueden modificarse, donde se ve que está configurado para que se muestran los últimos 30 minutos de mediciones y que se actualice el panel de control cada 5 segundos.
- **El cuadrado rojo**, indica el aforo actual del espacio cerrado donde se encuentra la Raspberry Pi con cámara y el acelerador USB Coral.
- **El rectángulo verde**, a la izquierda de la figura 5.34, muestra los niveles de CO2 en ppm (partes por millón), medidos por dos ESP32 que se encuentran desplegados en el espacio cerrado de mi habitación.
- **El rectángulo amarillo**, a la derecha de la figura 5.34, muestra la temperatura en grados centigrados, medida por los dos ESP32 que se encuentran desplegados en el espacio cerrado de mi habitación.

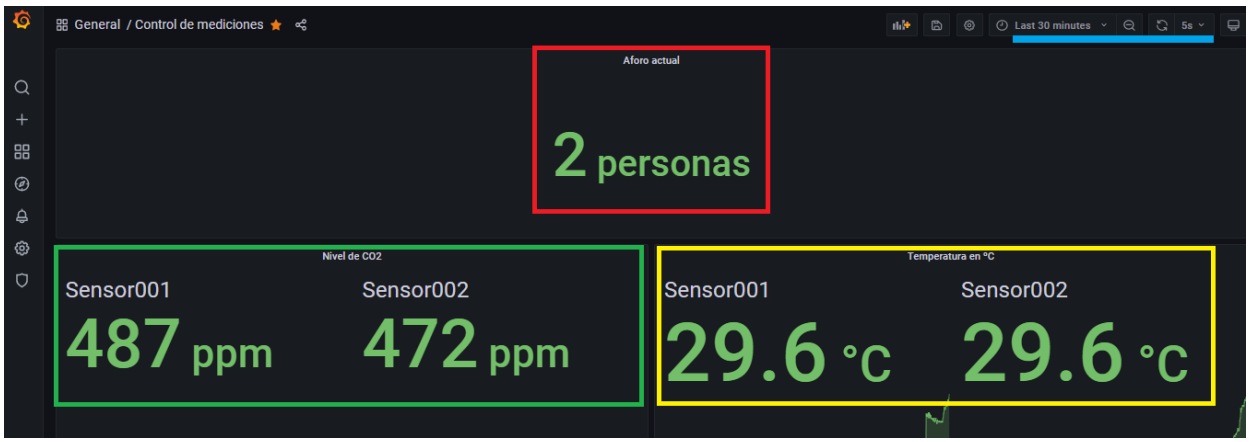


Figura 5.34 Panel de control de Grafana explicativo

En la figura 5.35 se muestra la evolución de las mediciones de CO2, TVOC, temperatura y humedad, tomados por el ESP32 con nombre "Sensor001".

Figura 5.35 Panel de control donde se muestra, a lo largo del tiempo, la evolución de las mediciones de CO2, TVOC, temperatura y humedad

Como ya se comentó, estos paneles son simples ejemplos de cómo se puede visualizar la información de una forma amigable y entendible para cualquier persona utilizando Grafana.

5.5.2. Mensajes del bot de Telegram "@esp32_ucm_bot"

En este apartado se verán los resultados de la interacción entre un usuario y el bot de Telegram encargado de gestionar las suscripciones y de emitir las alarmas cuando se sobrepasan los límites de los niveles configurados en Node Red.

5.5.2.1. Gestión de las suscripciones en el bot

A continuación se mostrarán las figuras 5.36, 5.37, 5.38, 5.39, 5.40 y 5.41, donde se verá la gestión e interacción entre usuario y bot.

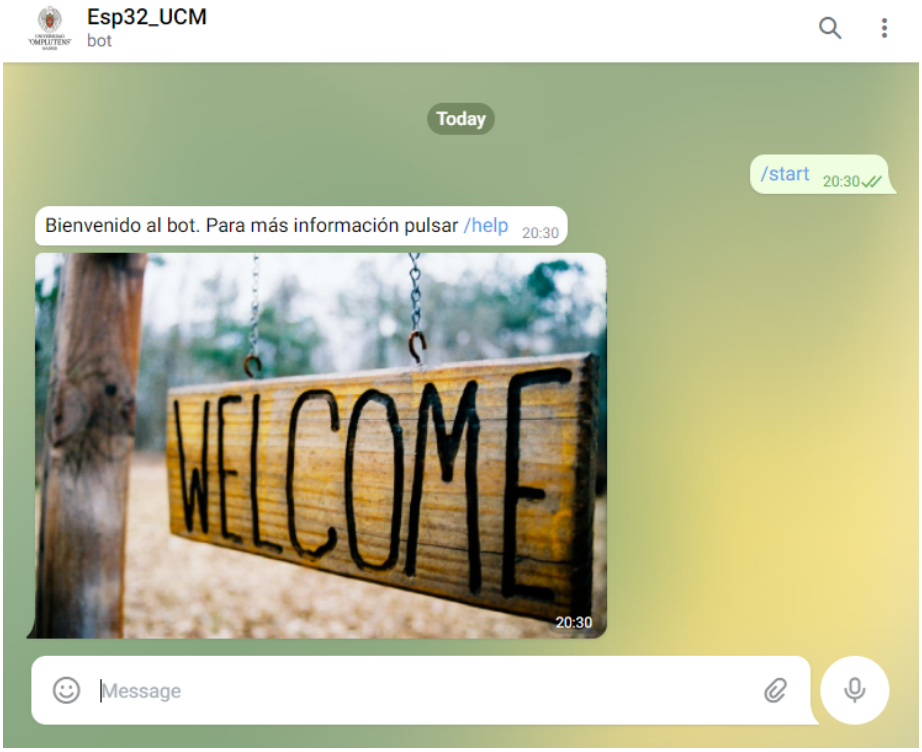


Figura 5.36 Respuesta del bot cuando se teclea "/start"

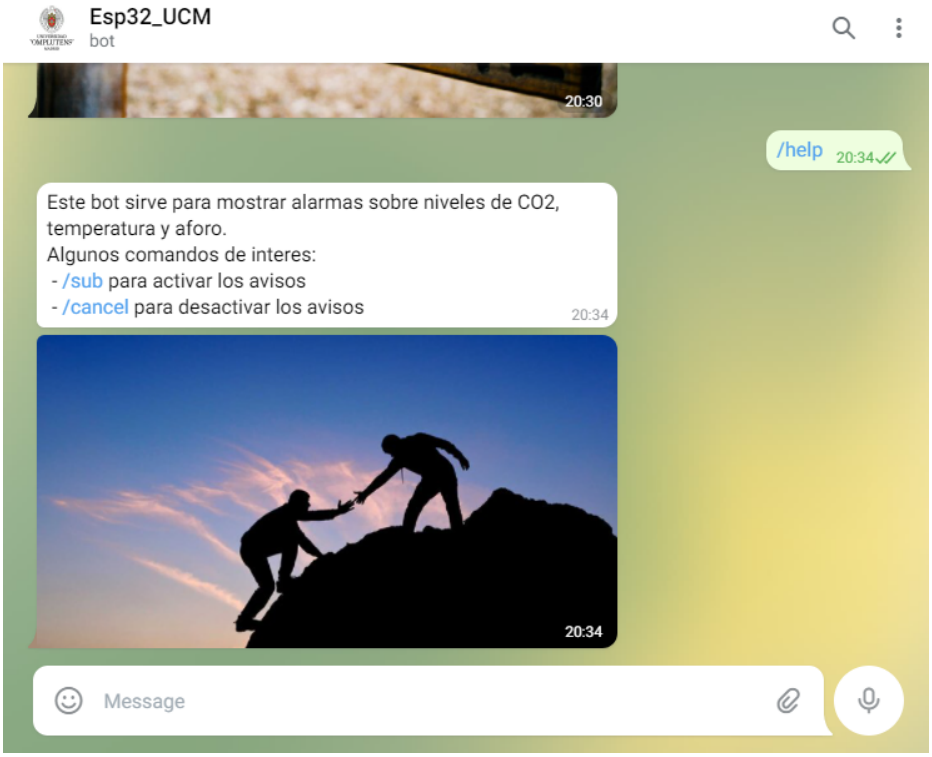


Figura 5.37 Respuesta del bot cuando se teclea "/help"

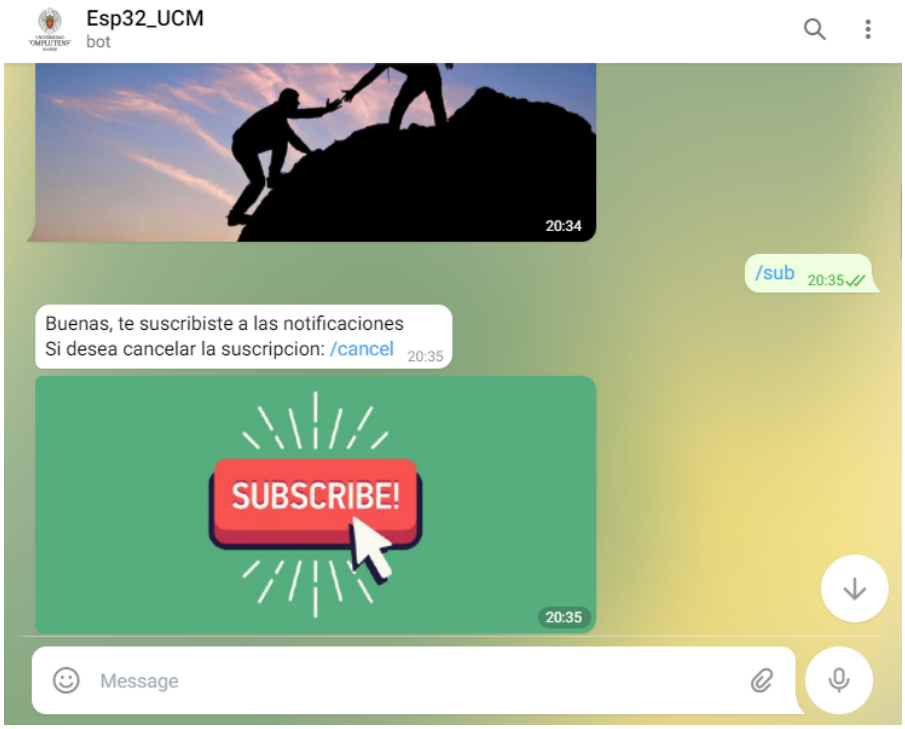


Figura 5.38 Respuesta del bot cuando se teclea "/sub" y no se está suscrito

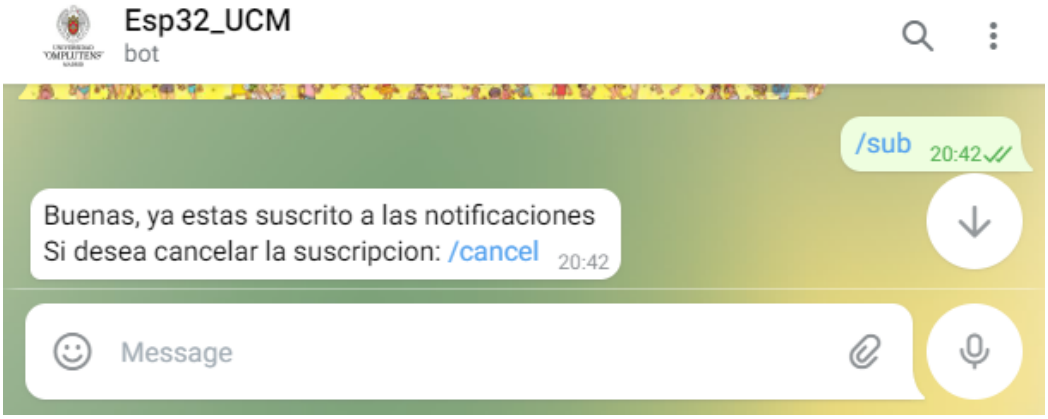


Figura 5.39 Respuesta del bot cuando se teclea "/sub" y ya se estaba suscrito

Figura 5.40 Respuesta del bot cuando se teclea "/cancel" y se está suscrito

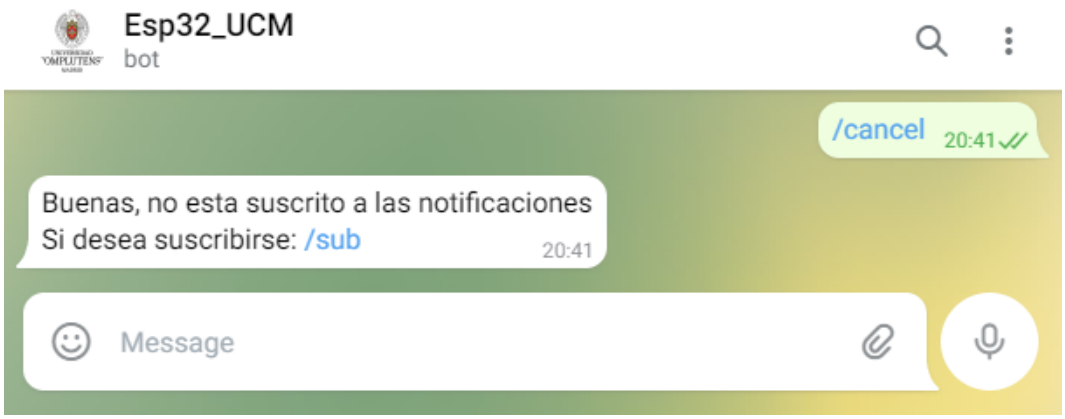


Figura 5.41 Respuesta del bot cuando se teclea "/cancel" y no se estaba suscrito

5.5.2.2. Notificación de las diferentes alarmas (CO2, temperatura y aforo)

A continuación se mostrarán las figuras 5.42, 5.43 y 5.44, donde se verán las alarmas que envía el bot de Telegram a los usuarios suscritos.

A modo de demostración se ajustarán los niveles máximos permitidos para facilitar que las alarmas salten, los niveles máximos para la demostración se pueden ver en la Tabla 5.1.

Medición	**Nivel máximo**
CO2	500 ppm
Temperatura	20 ºC
Aforo	1 persona

Tabla 5.1 Niveles máximos para la demostración

Figura 5.42 Alarma que avisa de que se sobrepasó el límite del CO2

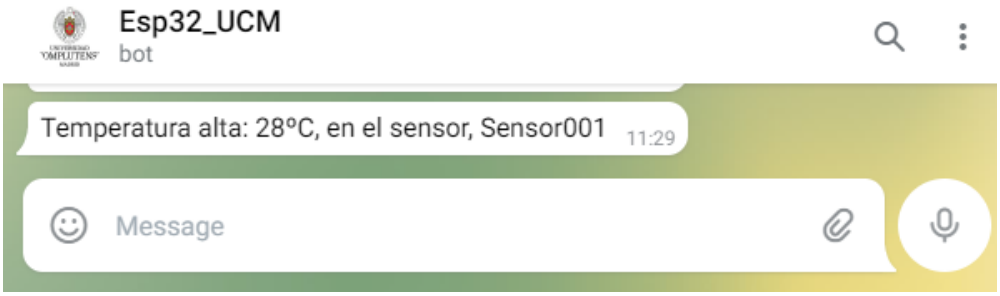


Figura 5.43 Alarma que avisa de que se sobrepasó el límite de temperatura

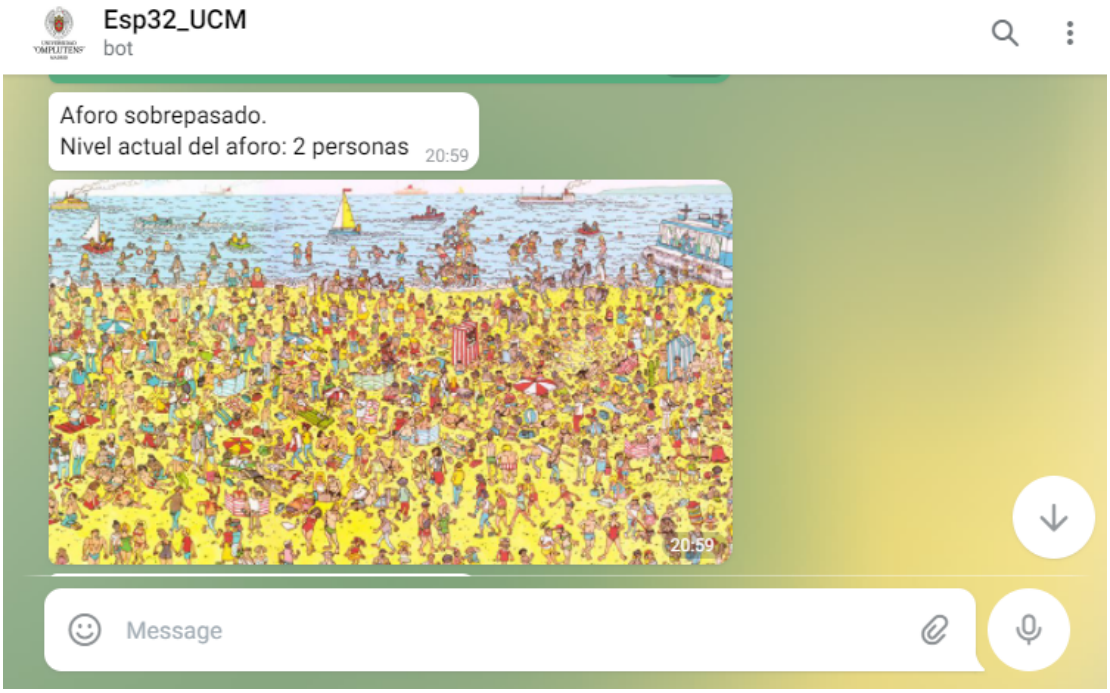


Figura 5.44 Alarma que avisa de que se sobrepasó el límite del aforo

7. Conclusiones y trabajo futuro

7.1. Conclusiones

En lo que respecta a los objetivos planteados al inicio del proyecto, se han cumplido y alcanzado todos ellos, consiguiendo implementar un sistema compuesto por diferentes dispositivos, pudiendo tomar mediciones, enviarlas por Internet de forma encriptada, procesarlas, almacenarlas y visualizarlas de forma amigable e informativa. De esta manera se demostró el potencial del concepto IoT, aplicado a la medición de la calidad del aire de manera remota dentro de un espacio cerrado y con la capacidad de poder notificar aquellas mediciones que sobrepasen ciertos límites, previamente configurados.

Entre los dispositivos utilizados se encuentra el microcontrolador ESP32 capaz de hablar con los sensores SGP30 y Si7021, encargados de medir los niveles de CO2, TVOC, temperatura y humedad. Estos datos serán enviados por Wifi utilizando cifrado, para una mayor privacidad y seguridad en las comunicaciones. Aparte, el ESP32 tendrá la capacidad de saber la zona horaria donde se encuentra y poder gestionar el consumo energético, proporcionando un plus de ahorro en el consumo cuando no sea necesario su uso.

Por otro lado, se encuentra la Raspberry Pi con cámara, capaz de medir el aforo aproximado en un espacio cerrado, mediante el uso del modelo de detección de objetos SSD-MobileNet v2 y el código proporcionado por Google Coral, que fue adaptado a las necesidades del proyecto, para que se centrará en detectar personas y que pudiera transmitir el aforo aproximado utilizando cifrado.

Las mediciones enviadas de forma encriptada por el ESP32 y la Raspberry Pi con cámara, serán recibidas por el mini servidor en Raspberry Pi, que será el encargado de procesar, almacenar y visualizar toda la información que tiene que ver con la medición de la calidad del aire en un espacio cerrado. Para ello se usará la tecnología Docker, que permite tener separados los datos de los diferentes servicios utilizados en el proyecto. De esta manera se facilita su portabilidad y se evitan problemas de dependencias en la instalación. También, se desarrolló un sistema de gestión de alarmas para enviar avisos a través de un bot en Telegram a los usuarios suscritos y poder informar sobre el empeoramiento de la calidad del aire en un espacio cerrado.

Este sistema dará al usuario el poder de controlar los datos generados de las mediciones, sin tener que pagar suscripciones por usar los servicios de procesamiento, almacenamiento y visualización, lo que evitará recurrir a los servicios ofrecidos por AWS y Microsoft Azure, entre otros.

7.2. Trabajo futuro

A continuación se pasará a citar las posibles ampliaciones para este proyecto:

- Faltaría realizar pruebas de campo, para ver cómo se desenvuelve el sistema en un entorno real, como pueda ser un aula o la sala de un cine.
- Situar la Raspberry Pi con cámara en una posición estratégica dentro del espacio cerrado, como puertas de entrada y salida, haciendo uso de los "bounding boxes"(son las coordenadas de un objeto detectado dentro de una imagen) y combinar el tracking de Google Coral, para poder identificar a un objeto asignándole un ID hasta que este salga de la imagen. De esta forma cuando entre o salga una persona se podrá mejorar el cálculo del aforo.
- Estudiar otros modelos de detección de objetos como SSDLite-MobileDet o EfficientDet-Lite, para comparar con el modelo SSD-MobileNet v2 y evaluar cual es el modelo que mejor se adapta para la detección de personas.
- Mejorar el sistema de alertas del bot de Telegram para poder dar la opción al usuario de elegir a qué sensor suscribirse o desuscribirse.
- Implementar otros sistemas para notificar a los usuarios la calidad del aire, como enviar un correo electrónico o escribir un aviso en alguna red social, por ejemplo Twitter Tweet en una cuenta oficial de Twitter para interactuar con las personas.

8. Introduction

1.1. Motivation

The IoT or Internet of Things is the real-time digital interconnection of everyday devices with the Internet, which allows the collection and sharing of information with other devices or control centers.

To achieve data collection, sensors play a crucial role as they are embedded in all kinds of physical devices. In this way, they can be integrated within a mobile device, appliances, vehicles, traffic signs and almost any object that can be found on a day-to-day basis. These sensors generate large amounts of data, which will be transferred to IoT platforms for further processing or storing.

IoT platforms are responsible for processing, storing and analyzing large volumes of data to share the results and / or predictions. This allows offering a better user experience or making production processes more efficient.

The applications for IoT are almost limitless. Here are some of them:

- Smart farms, which allow the farmer to be more efficient in his crops, saving water with intelligent irrigation or driving the tractors autonomously.
- Smart factories, better known by the name of IIoT (Industrial Internet of Things), which will help to monitor products in real time, inventory management or control production flows.
- Smart home, where the ability to control household appliances with voice commands or from another device such as a smartphone will be given.

IoT is used to solve real life problems. A current problem is the context of a COVID-19 pandemic, caused by the emergence of a new virus that is transmitted through the air and has had a rapid spread throughout the world.

This project provides an IoT solution to reduce the spread of COVID-19, by controlling air quality in closed spaces, since air is the main means of spread of COVID-19 and, hence, a correct air quality can decrease the ability of its transmission in closed spaces.

1.2. Objectives

The main objective of the project is to develop a system capable of collecting, storing and displaying data from measurements of CO2, TVOC, temperature, humidity, and occupation, and hence to be able to measure air quality in an enclosed space. Also, the system will be able to send an alert to the mobile phone, if a certain level of CO2, temperature or occupation is exceeded.

In order to achieve this general goal, this main objective is divided into more specific objectives:

- To develop a piece of software that allows a sensor node to manage and send measurements of CO2, TVOC, temperature and humidity levels. In this way, it will be possible to infer the air quality in a closed space.
- To develop a system capable of measuring the current occupation of an enclosed space. The developed solution will be able to detect people through the use of an object detection model, by means of images captured by a camera.
- To develop a system capable of processing, storing and visualizing the data sent by the air quality measurement and the measurement of the occupation. Some of its specific functionalities will be:
    - Receiving the data sent by the sensor nodes (air quality and capacity meter), via MQTT with SSL / TLS transport, using the eclipse-mosquitto MQTT broker.
    - Managing the interactions between different applications using Node Red.
    - Stornig the large amount of data received in InfluxDB, this being a time series database.
    - Visualizing the stored data, in a control panel, using Grafana.
    - Storing in MongoDB the necessary information about the users who subscribe to the alerts in the Telegram bot to be able to send them alarms when certain limits are exceeded that indicate poor air quality.

1.3. Workplan

To achieve the above objectives, the following specific tasks were developed:

- A deep research on the ESP32 device. In this first task, information was sought about how to develop software to be able to send the data via MQTT with TLS encryption and how to communicate the ESP32 with the Si7021 and SGP30 sensors, to carry out the tasks of measuring the levels of CO2, TVOC, temperature and humidity.
- Research on software developed for object detection. Specifically, to detect people in an image, and to leverage this information for inferencing the amount of people sharing an enclosed space.
- Design and development of a system for data processing, storage and visualization of the gathered data. Information was sought about Docker and how this tool could help meet the needs of deploying services capable of processing, storing and visualizing data without causing compatibility problems in the installation of applications. This approach, in addition, allows it to be easily deployed on any device supporting Docker.
- Tests on the system. This final task consisted of carrying out different evaluations of the system, such as seeing if the control panel visualized the measurements provided by the sensors in real time and checking that it sent alert messages to the Telegram bot when certain levels of CO2, temperature were exceeded. or capacity.

9. Conclusions and future work

7.1. Conclusions

Regarding the objectives set at the beginning of the project, all of them have been met and achieved; we have implemented a complete infrastructure composed by different devices, being able to take measurements, send them over the Internet in an encrypted way, process, store and view them in a friendly and informative manner. In this way, the potential of the IoT concept was demonstrated, applied in our case to the measurement of air quality remotely within a closed space and with the ability of notifying those measurements that exceed certain limits, previously configured.

The ESP32 microcontroller is the core part of our development, being able to take measurements from the SGP30 and Si7021 sensors, responsible for measuring the levels of CO2, TVOC, temperature and humidity. These data will be transmitted via Wifi using encryption, for greater privacy and security in communications. In addition, the ESP32 features the ability of determining the time zone where it is located and to be able to manage energy consumption, providing an additional (and configurable) saving in consumption when its use is not strictly necessary.

A second actor in our deployment is a system that determines the current occupation of an enclosed space, running on a Raspberry Pi with a camera. This estimation is based on the use of the SSD-MobileNet v2 object detection model, which was adapted to the needs of the project, so that it will focus on detecting people and can transmit the approximate capacity using encryption.

The measurements sent in an encrypted form by the ESP32 and the Raspberry Pi with camera are received by a centralized server running on a second Raspberry Pi, which is in charge of processing, storing and displaying all the information related with measuring the quality of the air in an enclosed space. For this, Docker technology has been selected, which allows the data of the different services used in the project to be isolated, and facilitates the deployment and integration of components. In this way its portability is facilitated and dependency problems in the installation are avoided. Also, an alarm management system has been developed to send notifications through

Telegram bot to subscribed users and to be able to report on the worsening of air quality in a closed space.

This system gives the user the power to control the data generated from the measurements, without having to pay subscriptions to use the processing, storage and visualization services, which will avoid resorting to the services offered by AWS and Microsoft Azure, among others.

7.2. Future work

The possible extensions for this project will be cited below:

- It would be necessary to carry out field tests, to see how the system performs in a real environment, such as a classroom or a movie theater.

- Place the Raspberry Pi with camera in a strategic position within the closed space, such as entrance and exit doors, making use of the "bounding boxes" (they are the coordinates of an object detected within an image) and combine Google Coral tracking, to be able to identify an object by assigning it an ID until it exits the image. In this way, when a person enters or leaves, the calculation of the capacity can be improved.

- Study other object detection models such as SSDLite-MobileDet or EfficientDet-Lite, to compare with the SSD-MobileNet v2 model and to evaluate which model is best suited for the detection of people.

- Improve the Telegram bot's alert system to be able to give the user the option of choosing which sensor to subscribe or unsubscribe to.

- Implement other systems to notify users of air quality, such as sending an email or writing a notice on a social network, for example Twitter Tweet on an official Twitter account to interact with people.

Bibliografía

Aquí irán los enlaces que den soporte a la memoria

[1] PC componentes. (s. f.). Www.Pccomponentes.Com. Recuperado 6 de septiembre de 2021, de https://www.pccomponentes.com/phasak-jd-3002-medidor-co2-negro#:%7E:text=Gracias%20al%20sensor%20digital%20incorporado,sobre%20la%20calidad%20del%20aire.&text=Equipado%20con%20la%20funci%C3%B3n%20de,aire%20no%20sea%20la%20adecuada

[2] W. (2021, 11 mayo). Consigue tu Medidor de CO2 para la Calidad del Aire. Blog. Noticias y Actualidad | Rodavigo Suministros Industriales. https://rodavigo.net/blog/consigue-medidor-co2-calidad-del-aire/

[3] D. (s. f.). GitHub - dusty-nv/jetson-inference: Hello AI World guide to deploying deep-learning inference networks and deep vision primitives with TensorRT and NVIDIA Jetson. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/dusty-nv/jetson-inference#inference

[4] G. (s. f.-b). GitHub - google-coral/example-object-tracker. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/google-coral/example-object-tracker

[5] colaboradores de Wikipedia. (2021, 19 agosto). Microsoft Azure. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Microsoft_Azure

[6] Wikipedia contributors. (2021, 3 agosto). ThingSpeak. Wikipedia. https://en.wikipedia.org/wiki/ThingSpeak

[7] colaboradores de Wikipedia. (2021a, febrero 15). I2C. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/I%C2%B2C

[8] Getting Started - CJSON - DocsForge. (s. f.). Cjson.Docsforge.Com. Recuperado 6 de septiembre de 2021, de https://cjson.docsforge.com/master/getting-started/

[9] Power Management - ESP32 - — ESP-IDF Programming Guide latest documentation. (s. f.). Docs.Espressif.Com. Recuperado 6 de septiembre de

2021, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/power_management.html#configuration

[10] Wi-Fi & Bluetooth MCUs and AIoT Solutions I Espressif Systems. (s. f.). www.espressif.com. Recuperado 6 de septiembre de 2021, de https://www.espressif.com/

[11] Get Started - ESP32 - — ESP-IDF Programming Guide latest documentation. (s. f.). Docs.Espressif.Com. Recuperado 6 de septiembre de 2021, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#step-1-install-prerequisites

[12] Espressif. (s. f.-b). *GitHub - espressif/esp-idf: Espressif IoT Development Framework. Official development framework for ESP32*. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/espressif/esp-idf

[13] *API Reference - ESP32 - — ESP-IDF Programming Guide latest documentation*. (s. f.). Documentación ESP32 - Espressif. Recuperado 6 de septiembre de 2021, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html

[14] *ESP32 Forum - Index page*. (s. f.-b). ESP32 - FORUM. Recuperado 6 de septiembre de 2021, de https://www.esp32.com/

[15] colaboradores de Wikipedia. (2021b, abril 22). Sensor de efecto Hall. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Sensor_de_efecto_Hall

[16] colaboradores de Wikipedia. (2021a, enero 5). I2S. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/I%C2%B2S

[17] colaboradores de Wikipedia. (2021d, junio 27). Serial Peripheral Interface. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Serial_Peripheral_Interface

[18] colaboradores de Wikipedia. (2021a, enero 5). Bus CAN. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Bus_CAN

[19] colaboradores de Wikipedia. (2020, 27 agosto). ESP32. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/ESP32

[20] ESP32 datasheet. (2021, julio). https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[21] ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems. (s. f.). www.espressif.com. Recuperado 6 de septiembre de 2021, de https://www.espressif.com/en/products/socs/esp32

[22] Engineers, L. M. (2020, 18 diciembre). Insight Into ESP32 Sleep Modes & Their Power Consumption. Last Minute Engineers. https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

[23] espressif. (s. f.-a). esp-idf/examples/protocols/mqtt/ssl at master · espressif/esp-idf. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/espressif/esp-idf/tree/master/examples/protocols/mqtt/ssl

[24] espressif. (s. f.-c). esp-idf/examples/protocols/sntp at 64b56beff5a4d6a165bd88027f585bfe51fbb990 · espressif/esp-idf. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/espressif/esp-idf/tree/64b56beff5a4d6a165bd88027f585bfe51fbb990/examples/protocols/sntp

[25] espressif. (s. f.-a). esp-idf/examples/peripherals/i2c/i2c_self_test/main at master · espressif/esp-idf. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/espressif/esp-idf/tree/master/examples/peripherals/i2c/i2c_self_test/main

[26] espressif. (s. f.-f). esp-idf/sntp_example_main.c at 178b122c145c19e94ac896197a3a4a9d379cd618 · espressif/esp-idf. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/espressif/esp-idf/blob/178b122c145c19e94ac896197a3a4a9d379cd618/examples/protocols/sntp/main/sntp_example_main.c

[27] A. (2019, 31 mayo). I2C - Puerto, Introducción, trama y protocolo. HETPRO/TUTORIALES. https://hetpro-store.com/TUTORIALES/i2c/

[28] Sensirion_Gas_Sensors_SGP30_Datasheet_EN. (2017, agosto). https://cdn-learn.adafruit.com/assets/assets/000/050/058/original/Sensirion_Gas_Sensors_SGP30_Datasheet_EN.pdf

[29] Support_Documents_TechnicalDocs_Si7021-A20. (2016, agosto). https://cdn-learn.adafruit.com/assets/assets/000/035/931/original/Support_Documents_TechnicalDocs_Si7021-A20.pdf

[30] S. (2020, 21 octubre). Cómo medir la calidad del aire. Soloelectronicos.com. https://soloelectronicos.com/2020/10/21/como-medir-la-calidad-del-aire/

[31] MQTT - The Standard for IoT Messaging. (s. f.-b). Mqtt.Org. Recuperado 6 de septiembre de 2021, de https://mqtt.org/

[32] L. (2019b, abril 17). ¿Qué es MQTT? Su importancia como protocolo IoT. Luis Llamas. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

[33] Vea nuestro seminario web para comprender la función y los componentes de mensajería en el diseño de aplicaciones. (s. f.). Amazon Web Services, Inc. Recuperado 6 de septiembre de 2021, de https://aws.amazon.com/es/message-queue/

[34] OASIS Open. (2021, 29 enero). About Us. https://www.oasis-open.org/org/

[35] make-keys-docker-compose.sh. (s. f.). Gist. Recuperado 6 de septiembre de 2021, de https://gist.github.com/suru-dissanaike/fbb01a23cf9a138973732e76999c0d48

[36] Dissanaike, S. (2021, 29 mayo). Secure MQTT broker (TLS) and Docker Compose - himinds. Medium. https://medium.com/himinds/mqtt-broker-with-secure-tls-and-docker-compose-708a6f483c92

[37] colaboradores de Wikipedia. (2021f, julio 14). JSON. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/JSON

[38] Get Started - ESP32 - — ESP-IDF Programming Guide latest documentation. (s. f.-b). Docs.Espressif.Com. Recuperado 6 de septiembre de 2021, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/#step-1-install-prerequisites

[39] Project Configuration - ESP32 - — ESP-IDF Programming Guide latest documentation. (s. f.). Docs.Espressif.Com. Recuperado 6 de septiembre de 2021, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/kconfig.html#project-configuration

[40] Craggs, I. (s. f.). Eclipse Paho | The Eclipse Foundation. Www.Eclipse.Org. Recuperado 6 de septiembre de 2021, de https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php

[41] COCO - Common Objects in Context. (s. f.). cocodataset.org. Recuperado 6 de septiembre de 2021, de https://cocodataset.org/#download

[42] MobileNets: Open-Source Models for Efficient On-Device Vision. (2017, 14 junio). Google AI Blog. https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

[43] MobileNetV2: The Next Generation of On-Device Computer Vision Networks. (2018, 3 abril). Google AI Blog. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

[44] Howard, A. G. (2017, 17 abril). MobileNets: Efficient Convolutional Neural Networks for Mobile. . . ArXiv.Org. https://arxiv.org/abs/1704.04861v1

[45] Hollemans, M. (s. f.). Convolutional neural networks on the iPhone with VGGNet. Machinethink.Net. Recuperado 6 de septiembre de 2021, de https://machinethink.net/blog/convolutional-neural-networks-on-the-iphone-with-vggnet/

[46] Hollemans, M. (s. f.-b). MobileNet version 2. Machinethink.Net. Recuperado 6 de septiembre de 2021, de https://machinethink.net/blog/mobilenet-v2/

[47] How single-shot detector (SSD) works? | ArcGIS Developer. (s. f.). Developers.Arcgis.Com. Recuperado 6 de septiembre de 2021, de https://developers.arcgis.com/python/guide/how-ssd-works/

[48] Forson, E. (2019, 9 junio). Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning. Medium. https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

[49] Redmon, J. (2015, 8 junio). You Only Look Once: Unified, Real-Time Object Detection. ArXiv.Org. https://arxiv.org/abs/1506.02640

[50] Liu, W. (2015, 8 diciembre). SSD: Single Shot MultiBox Detector. ArXiv.Org. https://arxiv.org/abs/1512.02325

[51] Hui, J. (2020, 15 diciembre). SSD object detection: Single Shot MultiBox Detector for real-time processing. Medium. https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06

[52] Hassan, M. U. (2021, 24 febrero). VGG16 – Convolutional Network for Classification and Detection. Neurohive. https://neurohive.io/en/popular-networks/vgg16/#:%7E:text=VGG16%20is%20a%20convolutional%20neural%20network%20model%20proposed%20by%20K.&text=Zisserman%20from%20the%20University%20of,images%20belonging%20to%201000%20classes

[53] USB Accelerator. (s. f.). Coral. Recuperado 6 de septiembre de 2021, de https://coral.ai/products/accelerator/#documentation

[54] TensorFlow Lite | ML for Mobile and Edge Devices. (s. f.). TensorFlow. Recuperado 6 de septiembre de 2021, de https://www.tensorflow.org/lite

[55] The Raspberry Pi Foundation. (s. f.). Buy a Camera Module 2 –. Raspberry Pi. Recuperado 6 de septiembre de 2021, de https://www.raspberrypi.org/products/camera-module-v2/

[56] IMX219 datasheet and driver - Raspberry Pi Forums. (s. f.). Www.Raspberrypi.Org. Recuperado 6 de septiembre de 2021, de https://www.raspberrypi.org/forums/viewtopic.php?t=177308

[57] colaboradores de Wikipedia. (2021g, agosto 11). Docker (software). Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Docker_(software)

[58] Villacampa, Ó. (2021, 29 enero). Qué es Docker y para qué sirve. Ondho. https://www.ondho.com/que-es-docker-para-que-sirve/

[59] colaboradores de Wikipedia. (2020b, septiembre 11). Portainers (Docker). Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Portainers_(Docker)

[60] Installing Docker Compose. (2019, 22 junio). YouTube. https://www.youtube.com/watch?v=rALEXTZedjI

[61] Docker Hub Quickstart. (2021, 2 septiembre). Docker Documentation. https://docs.docker.com/docker-hub/#:%7E:text=Docker%20Hub%20is%20a%20service,container%20images%20with%20your%20team.&text=Docker%20Hub%20provides%20the%20following,private%20repositories%20of%20container%20images

[62] Solar, D. (2021, 27 enero). Portainer - Gestor de contenedores Docker. YouTube. https://www.youtube.com/watch?t=212&v=_vvpLnvmITg&feature=youtu.be

[63] Docker Hub. (s. f.). Hub.Docker.Com. Recuperado 6 de septiembre de 2021, de https://hub.docker.com/

[64] Docker Hub. (s. f.-b). Hub.Docker.Com. Recuperado 6 de septiembre de 2021, de https://hub.docker.com/r/portainer/portainer-ce

[65] Wikipedia contributors. (2021b, septiembre 4). InfluxDB. Wikipedia. https://en.wikipedia.org/wiki/InfluxDB

[66] InfluxData. (2021, 28 agosto). InfluxDB: Purpose-Built Open Source Time Series Database. https://www.influxdata.com/

[67] Wikipedia contributors. (2021b, agosto 29). MIT License. Wikipedia. https://en.wikipedia.org/wiki/MIT_License

[68] Wikipedia contributors. (2021a, junio 21). Implied warranty. Wikipedia. https://en.wikipedia.org/wiki/Implied_warranty#Disclaimer_of_an_implied_warranty

[69] colaboradores de Wikipedia. (2021i, agosto 31). Go (lenguaje de programación). Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Go_(lenguaje_de_programaci%C3%B3n)

[70] R., & R. (s. f.-c). docker_compose_cookbook/docker-compose.yml at master · robcowart/docker_compose_cookbook. GitHub. Recuperado 6 de septiembre de 2021, de https://github.com/robcowart/docker_compose_cookbook/blob/master/STACKS/influx_oss/docker-compose.yml

[71] Habilitar autenticación para HTTP en InfluxDB. (2018, 19 octubre). Linuxito. https://www.linuxito.com/seguridad/1118-habilitar-autenticacion-para-http-en-influxdb

[72] Docker Hub. (s. f.-c). Hub.Docker.Com. Recuperado 6 de septiembre de 2021, de https://hub.docker.com/_/influxdb

[73] colaboradores de Wikipedia. (2021g, julio 30). Grafana. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Grafana

[74] Grafana: The open observability platform. (s. f.). Grafana Labs. Recuperado 6 de septiembre de 2021, de https://grafana.com/

[75] colaboradores de Wikipedia. (2021c, febrero 6). Apache License. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Apache_License

[76] Overview of Docker Compose. (2021, 2 septiembre). Docker Documentation. https://docs.docker.com/compose/

[77] Docker Hub. (s. f.-d). Hub.Docker.Com Grafana. Recuperado 6 de septiembre de 2021, de https://hub.docker.com/r/grafana/grafana

[78] colaboradores de Wikipedia. (2021, 2 julio). MongoDB. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/MongoDB

[79] colaboradores de Wikipedia. (2019, 12 octubre). BSON. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/BSON

[80] colaboradores de Wikipedia. (2020, 8 junio). GNU Affero General Public License. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/GNU_Affero_General_Public_License

[81] Patel, K. (2020, 28 agosto). How to Run MongoDB on Your Local Network Using a Raspberry Pi and Docker. Medium. https://medium.com/swlh/how-to-run-mongodb-on-local-network-using-a-raspberry-pi-and-docker-4e5c4379cea2

[82] Λ. (2020, 22 enero). Enable mongodb authentication with docker - Rahasak Labs. Medium. https://medium.com/rahasak/enable-mongodb-authentication-with-docker-1b9f7d405a94

[83] docker-library. (s. f.). User is not created during execution of init script · Issue #399 · docker-library/mongo. GitHub. Recuperado 7 de septiembre de 2021, de https://github.com/docker-library/mongo/issues/399

[84] Docker Hub. (s. f.). Hub.Docker.Com MongoDB. Recuperado 7 de septiembre de 2021, de https://hub.docker.com/_/mongo?tab=description&page=1&ordering=last_updated&name=bionic

[85] Eclipse Mosquito. (s. f.). mosquitto.org. Recuperado 7 de septiembre de 2021, de https://mosquitto.org/

[86] colaboradores de Wikipedia. (2019a, septiembre 23). Eclipse Public License. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Eclipse_Public_License

[87] Dissanaike, S. (2021, 29 mayo). Secure MQTT broker (TLS) and Docker Compose - himinds. Medium. https://medium.com/himinds/mqtt-broker-with-secure-tls-and-docker-compose-708a6f483c92

[88] Docker Hub. (s. f.-b). Hub.Docker.Com Eclipse Mosquito. Recuperado 7 de septiembre de 2021, de https://hub.docker.com/_/eclipse-mosquitto

[89] Node-RED. (s. f.). Nodered.Org. Recuperado 7 de septiembre de 2021, de https://nodered.org/

[90] Wikipedia contributors. (2021, 3 septiembre). Apache License. Wikipedia. https://en.wikipedia.org/wiki/Apache_License#Version_2.0

[91] Training, I. S. A. (2021, 24 enero). Adding Nodes to Node Red. YouTube. https://www.youtube.com/watch?t=44&v=t2oOKcEPYYA&feature=youtu.be

[92] R. (2019, 1 noviembre). Raspberry Pi Raspbian Setup Configuration Using Graphical Interface. YouTube. https://www.youtube.com/watch?t=133&v=rDbBlaAn8E8&feature=youtu.be

[93] Docker Hub. (s. f.-c). Hub.Docker.Com Node Red Docker. Recuperado 7 de septiembre de 2021, de https://hub.docker.com/r/nodered/node-red-docker/

[94] node-red-contrib-telegrambot. (s. f.). Flows.Nodered.Org Telegrambot. Recuperado 7 de septiembre de 2021, de https://flows.nodered.org/node/node-red-contrib-telegrambot

[95] colaboradores de Wikipedia. (2021b, septiembre 2). Telegram. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Telegram

[96] Install Docker Engine on Ubuntu. (2021, 2 septiembre). Docker Documentation. https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script

Apéndices

Apéndice I - Código del proyecto

El código del proyecto estará disponible en el siguiente enlace de Google Drive:

https://drive.google.com/drive/folders/19r7Zs6enmU9MaexusTom1Tc2DAAnRfFX?usp=sharing